



*Ariel
Hardware Reference
Library*

EDX/J Language and Supervisor Description

Z109-0115-00



*Ariel
Hardware Reference
Library*

EDX/J Language and Supervisor Description

EDX/J Language and Supervisor Description

Control Systems Engineering

IBM Corporation
General Technology Division
D-035 B-300-47A
East Fishkill
tel 533-7654

IBM Internal Use Only

Z109-0115-00

First Edition April 1987

Changes to this information will be published in revisions to this edition.

Published By: Manufacturing Engineering, D541 Toronto

Z109-0115-00

IBM Internal Use Only

IBM Internal Use Only

TABLE OF CONTENTS

1.0	Introduction	1
2.0	Summary of Changes	3
2.1.1.1	Revision 7 - September 1986	3
3.0	System Overview	5
3.1	General Description	5
3.2	System Components	6
3.3	Summary of Features	7
3.4	Minimum Practical System Configuration	8
3.5	Programming Systems	9
4.0	Application Programming User's Guide	11
4.1	Introduction	11
4.2	General Command Format	12
4.2.1	Address Indexing Feature	14
4.2.2	Use of the Parameter Naming Operand Px=	16
4.2.3	Symbolic I/O Assignments	17
4.2.4	Data Movement, Logical and Arithmetic Functions	18
4.2.5	Derived Floating Point Functions	22
4.2.6	Floating Point Arithmetic Instructions	23
4.2.7	Task Control	25
4.2.8	Program Sequencing	26
4.2.9	Terminal Input/Output	28
4.2.10	Host Communication Functions	30
4.2.11	Disk Functions	31
4.2.12	Listing Control Instructions	32
4.3	Command Descriptions	33
4.3.1	ADD	33
4.3.2	ADDV	34
4.3.3	AND	35
4.3.4	ATTACH	36
4.3.5	ATTNLIST	37
4.3.6	BUFFER	38
4.3.7	CALL	39
4.3.8	CANCEL	40
4.3.9	CHKEY	41
4.3.10	CLOSE	42
4.3.11	CONVTB	43
4.3.12	CONVTD	45
4.3.13	DATA	47
4.3.14	DEQ	49
4.3.15	DEQT	50
4.3.16	DEST	51
4.3.17	DETACH	52
4.3.18	DISIO - ARIEL	53
4.3.19	DISREAD - DSC	54
4.3.20	DISWRITE - DSC	57
4.3.21	DISREAD/DISWRITE - ARIEL	60
4.3.22	DIVIDE	62
4.3.23	DO	63
4.3.24	DSCB	65
4.3.25	ECB	66
4.3.26	EJECT	67
4.3.27	ELSE	68
4.3.28	END	69
4.3.29	ENDATTN	70
4.3.30	ENDDO	71
4.3.31	ENDIF	72
4.3.32	ENDPROG	73
4.3.33	ENDSELECT	74
4.3.34	ENDTASK	75
4.3.35	ENDTP	76
4.3.36	ENQ	77
4.3.37	ENQT	78
4.3.38	EOR	79
4.3.39	EQU	80
4.3.40	EXP	81
4.3.41	FADD	82
4.3.42	FDIVD	83

IBM Internal Use Only

4.3.43	FMULT	84
4.3.44	FPCONV	85
4.3.45	FREEMAIN	86
4.3.46	FSUB	87
4.3.47	GETMAIN	88
4.3.48	GETTIME	89
4.3.49	GETVALUE	90
4.3.50	GOTO	92
4.3.51	IF	93
4.3.52	INTBIT - DSC	95
4.3.53	INTBIT - ARIEL	96
4.3.54	INTIME	97
4.3.55	IOCB	98
4.3.56	IOR	99
4.3.57	LEAVE	100
4.3.58	LN	101
4.3.59	LOAD	102
4.3.60	LOADH	104
4.3.61	LOG	106
4.3.62	LSA - DSC	107
4.3.63	LSA - ARIEL	109
4.3.64	MAPAD	111
4.3.65	MDCB - ARIEL	112
4.3.65.1	DIS Error Log	113
4.3.65.2	DIS Internal Commands	114
4.3.65.3	DIS Internal Command Buffers	115
4.3.66	MOVE	118
4.3.67	MOVEA	119
4.3.68	MULTIPLY	120
4.3.69	OPEN	121
4.3.70	OTHER	122
4.3.71	POST	123
4.3.72	PRINDATE	124
4.3.73	PRINT	125
4.3.74	PRINTTEXT	126
4.3.75	PRINTIME	128
4.3.76	PRINTNUM	129
4.3.77	PROGRAM	131
4.3.78	PROGSTOP	133
4.3.79	PWR	134
4.3.80	QCB	135
4.3.81	QUESTION	136
4.3.82	READ	137
4.3.83	READTEXT	139
4.3.84	RECEIVE	141
4.3.85	RESET	143
4.3.86	RETURN	144
4.3.87	SELECT	145
4.3.88	SEND	148
4.3.89	SHIFTL	150
4.3.90	SHIFTR	151
4.3.91	SPACE	152
4.3.92	SQRT	153
4.3.93	STIMER	154
4.3.94	SUBROUT	155
4.3.95	SUBTRACT	156
4.3.96	SYSDEF	157
4.3.97	SYSOPEN	159
4.3.98	TASK	160
4.3.99	TCBGET	161
4.3.100	TEXT	162
4.3.101	TIME	163
4.3.102	TITLE	164
4.3.103	USER	165
4.3.104	WAIT	167
4.3.105	WHEN	168
4.3.106	WHEREAS	169
4.3.107	WRITE	170
5.0	Supervisor Utility Functions	173
5.1	Native Support	173
5.2	Remote Support	176
6.0	HAZEL/3 and Hazel/4 - Mini Debug Overview	177

IBM Internal Use Only

6.1	Display LSRs and PSWs - PF1 or TEST REQ	178
6.1.1	PSWs	178
6.1.2	XR's	178
6.2	Display and Patch Memory	180
6.2.1	Viewing Memory	180
6.2.1.1	Specifying Memory Address	180
6.2.1.2	Scrolling	180
6.2.2	Patching Memory	180
6.3	Other Mini Debug Functions	181
6.3.1	Start/Stop EDX Trace - PF2	181
6.3.2	Repeat Stop JIB - PF3	181
6.3.3	Display LSR Chapter and Extended PSWs - PF5	182
6.3.4	Change Partition (Hazel/4) - PF6	182
6.3.5	EDX Stop on Address - PF7	183
6.3.6	Memory Dump to S/I Diskette - PF8/PF9	183
6.3.7	JIB Stop on Address - PF10, PF11 or PF12	183
7.0	HAZEL with AP/CSS - Debug Tool Overview	185
7.1	Command Entry	186
7.2	Arithmetic Expressions	187
7.3	Global Commands	188
7.4	Primary Commands	190
7.4.1	Primary Commands Menu	190
7.4.2	Display Current CPU's Memory	190
7.4.3	Display the Currently Selected CPU's Registers	190
7.4.3.1	Hazel	190
7.4.3.2	AP/CSS	191
7.4.4	Dump Memory to S/I	192
7.4.5	Display System Queues	193
7.4.6	Return to Previous Primary Subsystem	193
7.4.7	Display Global Settings	193
7.4.8	Quit Debug	194
7.4.9	Stop JIB	194
7.4.10	Stop EDX	195
7.4.11	Stop Titles	196
7.4.12	Redisplay Stop Information	196
7.4.13	HAZEL Soft IPL	196
7.4.14	Trace EDX	197
7.4.15	EDX Trace Size	197
7.4.16	View EDX Trace	197
7.5	Memory Secondary Commands	199
7.5.1	Memory Commands Menu	199
7.5.2	Patch Memory at Currently Displayed Location	199
7.5.3	Patch Memory at Following Last Patch	199
7.5.4	Patch Memory at a Specified Address	199
7.5.5	Patch Repeat Word in Memory	200
7.5.6	Index Through Memory	200
7.5.7	Search for Word in Memory	200
7.5.8	Transfer Memory	200
7.5.9	Scroll through Memory	201
7.5.10	Reverse Scrolling Direction	201
7.6	Register Secondary Commands	202
7.6.1	Register Commands Menu	202
7.6.2	Select Registers by Chapter and Block	202
7.6.3	Display Register Titles	202
7.6.4	Patch Local Storage Registers	202
7.6.5	Patch HAZEL External Registers	203
7.6.6	Patch AP External Registers	203
7.7	Stop Secondary Commands	204
7.7.1	Stop Commands Menu	204
7.7.2	Single Step JIB/EDX	204
7.7.3	Go to User Specified Address	204
7.7.4	Patch EDX Index Registers	204
7.7.5	Display Memory	204
7.8	Miscellaneous Information	205
7.8.1	Warning Messages	205
7.8.2	Dates	205
7.8.2.1	The Assembly Date	205
7.8.2.2	The Last Modification Dates	205
7.8.3	Machine Check	205
7.8.3.1	Hazel	206
7.8.3.2	AP/CSS	206
7.8.3.3	AP/DISK	207
7.8.3.4	AP/DIS	207

IBM Internal Use Only

8.0	TRACES	209
8.1	Communications Trace - DSC	210
8.1.1	Communication Trace Buffer	210
8.1.2	Enabling/Disabling the Communication Trace	210
8.1.3	Communication Trace Layout	210
8.1.4	Communication Trace Layout Description	211
8.2	Communications Trace - S/I	214
9.0	Nucleus Generation	215
9.1	Nucleus Generation Commands	216
9.1.1	\$SYSCOM	216
9.1.2	COMM	218
9.1.3	CPU	219
9.1.4	DISK	220
9.1.5	ENDSYS	221
9.1.6	HDMRTST	222
9.1.7	NUCTYPE	223
9.1.8	SYSTEM	224
9.1.9	TERMINAL	225
10.0	Internal Design Guide	227
10.1	Introduction	227
10.2	Operating Conventions	227
10.2.1	Common Setup Routine	227
10.2.2	Register Conventions	227
10.2.3	Hardware Level Assignment of Hazel and APs	229
10.2.3.1	Hazel/3	229
10.2.3.2	Hazel/4 with Native Display	229
10.2.3.3	Hazel/4 with AP/CSS	229
10.2.3.4	Hazel/Ariel	229
10.2.3.5	AP/CSS	229
10.2.3.6	AP/DISK	230
10.2.3.7	AP/CSS - ARIEL	230
10.2.3.8	AP/DIS - ARIEL	230
10.3	Supervisor	231
10.3.1	General	231
10.3.2	Execution States	234
10.3.3	Task Synchronization and Control	235
10.3.3.1	EVENT Control Block (ECB)	235
10.3.3.2	Queue Control Block (QCB)	236
10.3.3.3	Task Control Block (TCB)	237
10.3.4	Functions	238
10.3.4.1	ATTACH	238
10.3.4.2	DETACH	238
10.3.4.3	WAIT	238
10.3.4.4	POST	238
10.3.4.5	ENQ	239
10.3.4.6	DEQ	239
10.3.5	TASK Dispatcher	240
10.3.5.1	General	240
10.3.5.2	Task Dispatcher Call	240
10.3.6	Supervisor Service Routines	242
10.3.6.1	\$CHAINP	242
10.3.6.2	REGSAVE	242
10.3.6.3	SVSETUP	242
10.3.6.4	LOADREG	242
10.3.7	Supervisor Work Area	243
10.3.8	Supervisor Function Calls	244
10.3.8.1	ATTACH	244
10.3.8.2	DETACH	244
10.3.8.3	POST	244
10.3.8.4	WAIT	245
10.3.8.5	DEQ	245
10.3.8.6	ENQ	245
10.3.9	Immediate Action - (IA) for Hardware level	247
10.3.10	Supervisor Call Routine - IA Level (Hardware level)	247
10.4	Interval Timer Support	248
10.4.1.1	General	248
10.4.1.2	Interval Timer Queue	248
10.4.1.3	Time Interval	248
10.4.1.4	Setting / Resetting the Interval Timer	248
10.5	Storage Map	250
10.6	Console - Terminal I/O Support	251
10.6.1	Organization	251

IBM Internal Use Only

10.6.2	Layer 1	252
10.6.2.1	Instruction Interpretation	252
10.6.2.2	Attention Handling	252
10.6.3	LAYER 2	253
10.6.3.1	Buffer Management	253
10.6.3.2	Numeric Conversion	253
10.6.4	LAYER 3	255
10.6.4.1	Device Support Routines	255
10.6.4.2	\$L5KYWT - Keyboard Wait Support	255
10.6.4.3	\$SCRNMGR - Screen Manager	255
10.6.4.4	\$DPYPROC - Display Processor	255
10.7	AP/CSS Console Support	257
10.7.1	Level 4 Interrupt Handler	260
10.7.2	Console Control Block (CCB)	261
10.8	Distributed Interface System (DIS)	265
10.8.1	General	265
10.8.2	MDB Description / Layout	266
10.8.3	MDCB Description / Layout	267
10.8.4	IDB Description / Layout	269
10.9	IPL Bring Up Tests / Initialization	270
10.9.1	Bring Up Tests - Hazel	270
10.9.2	Initialization	272
10.9.2.1	Hazel/3, Hazel/4 and Hazel/4 with AP.	272
10.9.2.2	Hazel/Ariel	272
10.9.3	Bring Up Tests - AP's	273
10.10	Communication Transmission Subsystem	275
10.10.1	Sub-System Requirements	275
10.10.2	Basic Protocol	275
10.10.2.1	Arbitration	275
10.10.2.2	Responses	275
10.10.2.3	Response Timeout	276
10.10.2.4	Transmit Timeout	276
10.10.2.5	Communication Modes	276
10.10.2.6	Stopping and Starting	276
10.10.3	Transmission Format	277
10.10.3.1	Control Format	277
10.10.3.2	Message Block Format	278
10.10.4	Awkward Situations	278
10.10.4.1	Tie Conditions	278
10.10.4.2	Receive Overflow	279
10.10.4.3	Handling Tie Conditions and Receive Overflow	279
10.10.4.4	Partially Full Receive Buffer	279
10.10.4.5	Handling Partially Full Receive Buffers	280
10.10.4.6	The Blast Reset Format	281
10.10.5	Interface to Communications Management	281
10.10.5.1	Send	281
10.10.5.2	Receive	281
10.10.6	Primary Entries in the Line Control Block	282
10.10.7	Protocol Examples	282
10.10.8	Program Loader	285
10.10.8.1	Request Block	285
10.10.8.2	Header Response Block	286
10.10.8.3	LOAD Sequence	286
10.10.9	Layout of the Communication Macros	287
10.11	DSC IPL LOAD	288
10.11.1	IPL Loader in Hazel	288
10.11.2	IPL Loader in APs	288
10.11.2.1	IPL Loader in AP/DISK	288
10.11.2.2	IPL Loader in AP/CSS	289
Appendix A. Syntax Summary		291
Appendix B. Equate Tables		293
B.1	Nucleus Directory and Equates	293
B.1.1	Nucleus Directory in Hazel/3	293
B.1.2	Nucleus Directory in Hazel/4 without AP/CSS	294
B.1.3	Nucleus Directory in Hazel/4 with AP/CSS	295
B.1.4	Nucleus Directory in Hazel/Ariel	296
B.2	OP Code Instruction Table	297
B.2.1	System Commands	297
B.2.2	Terminal Input/Output Control	298
B.2.3	Arithmetic Operations	298
B.2.4	Vector Addition	299
B.2.5	Logical Operations	299

IBM Internal Use Only

B.2.6	Loop Operations	300
B.2.7	Subroutine Organization	300
B.2.8	Branch Instructions	300
B.2.9	Timer Instructions	300
B.2.10	User Exit Instruction	300
B.2.11	Conversion Commands	300
B.2.12	Distributed Interface Channel I/O	300
B.2.13	APU Trig Support	301
B.2.14	ESF ODB Access Commands	301
B.2.15	ESF Stack Commands	301
B.2.16	TP Communication	301
B.2.17	Floating Point Arithmetic Commands	301
B.3	Task Control Block (TCB)	301
B.4	Console Control Block (CCB)	302
B.5	Macrofunction Data Control Block (MDCB)	303
B.6	Macrofunction Data Block (MDB)	304
B.7	Interrupt Data Block (IDB)	304
B.8	Program Header	304
Appendix C.	Character Sets	305
C.1	EBCDIC Character Set	305
C.2	ASCII Character Set	306
Appendix D.	Disk Directory Layout	307
Appendix E.	3101 Operation	309
Appendix F.	Hazel/4 - Hazel/Ariel DIS Support Compatibility	311
F.1	EDX Support for DIS	312
F.2	Compatibility	313
F.2.1	Syntax Compatibility	313
F.2.1.1	LSA	313
F.2.1.2	DISREAD and DISWRITE	313
F.2.2	Functional Compatibility	313
F.2.2.1	Internal Wait	313
F.2.2.2	Program Reference to MDB Fields	314
F.2.2.3	DIS I/O with COUNT=1	314
F.3	DIS Channel Performance	315
F.3.1	Measurement Tools	315
F.3.2	Results	316
Appendix G.	Memory Dump to S/1 Diskette	319
G.1	DISKETTE INITIALIZATION	320
G.1.1	\$DASDI and \$INITDSK	320
G.1.2	\$DISKUT1	320
G.2	DISABLING S/1 SENDS TO DSC	322
G.3	DSC MEMORY DUMP AND DISPLAY	323
G.3.1	STARTING DSC TO S/1 DUMP	323
G.3.1.1	DUMP PROCEDURE FOR RS232 BASED COMMUNICATIONS	323
G.4	TERMINATION / RESTART FOR RS232 BASED DUMP	324
G.4.1.1	DUMP PROCEDURE FOR AP/CSS BASED COMMUNICATIONS	324
G.4.1.2	TERMINATION / RESTART FOR AP/CSS BASED DUMP	324
G.4.2	DISPLAYING/UPLOADING THE DUMP	324
G.4.2.1	ALLOCATING DUMP DATA SET ON S/370	325
G.4.2.2	FORMATTING THE DUMP ON S/370	325
Appendix H.	S/1 COMMUNICATION TRACE	327
H.1	DISKETTE INITIALIZATION	328
H.1.1	Initializing the diskette	328
H.1.2	Creating the DUMP/TRACE dataset	328
H.2	USING THE TRACE FACILITY	330
H.2.1	Starting the Trace Programs	330
H.2.2	Added options for diskette tracing	331
H.2.2.1	DISPLAY command	331
H.2.3	Trace Overruns	332
H.2.3.1	Detection: P and C Options of DISPLAY	332
H.2.3.2	Dealing with the Problem	332
Appendix I.	S/1 COMPARE DATA SET UTILITY - \$COMP	335
I.1	Using \$COMP	336
I.1.1	Valid commands	336
I.1.1.1	The NF (New Files) command	336
I.1.1.2	The CD (Change Destination) command	336
I.1.1.3	The CA (Compare All) command	337

IBM Internal Use Only

I.1.1.4 The CP (Compare Partial) command	337
Appendix J. DISK UTILITY PROGRAM (DUT)	339
J.1 Dataset Qualification	340
J.2 Dataset Oriented Commands	341
J.3 File Oriented Commands	342
J.4 General Purpose Commands	344
Appendix K. S/I and DSC Utility Programs	345
Index	347

IBM Internal Use Only

LIST OF ILLUSTRATIONS

Figure 1.	A Simple EDX Program.	13
Figure 2.	Illustration of the Indexing Feature	14
Figure 3.	Register Dump Screen - Hazel/4	179
Figure 4.	Register Dump Screen - Hazel/3	179
Figure 5.	LSR Chapter and Extended PSWs Screen - Hazel/4	182
Figure 6.	Hazel Debug Primary Commands Menu.	190
Figure 7.	Hazel/4 Register Dump Screen.	191
Figure 8.	AP/CSS Register Dump Screen.	192
Figure 9.	Global Commands Menu and Current Settings.	194
Figure 10.	Memory Commands Menu.	199
Figure 11.	Register Commands Menu.	202
Figure 12.	Stop Commands Menu.	204
Figure 13.	AP/CSS Machine Check Register Dump - Hazel/Ariel.	206
Figure 14.	AP/CSS Machine Check Register Dump - Hazel/4.	206
Figure 15.	Module SUBSYS - Enable/Disable Communication Trace.	210
Figure 16.	Module COMTRACE - Enable/Disable Communication Trace.	210
Figure 17.	DSC Communication Trace Layout	211
Figure 18.	Sample Nucleus Generation Statements	215
Figure 19.	Supervisor Interface Structure	231
Figure 20.	Dispatcher Interface Structure	232
Figure 21.	System Control Flow for ATTACH	233
Figure 22.	System Control Flow with External Interrupt	233
Figure 23.	ECB Wait Chain with 3 Elements	235
Figure 24.	QCB Wait Chain with 3 Elements	236
Figure 25.	TCB Chain with 4 Sub-tasks	237
Figure 26.	Timer CSR Actions Part I	249
Figure 27.	Timer CSR Actions Part II	249
Figure 28.	Storage maps	250
Figure 29.	DCB Structure	258
Figure 30.	DCB Flag/Command Format	258
Figure 31.	AP Device Table	259
Figure 32.	PF Key Table	260
Figure 33.	CCB Native Display - I.	261
Figure 34.	CCB Native Display - II.	262
Figure 35.	CCB 3101 - I.	263
Figure 36.	CCB 3101 - II.	264
Figure 37.	MDB Format	266
Figure 38.	MDCB Format	267
Figure 39.	MDCB Flags and Commands	268
Figure 40.	IDB Format	269
Figure 41.	DBO Light Settings for Hazel/3 Bringup Errors	270
Figure 42.	DBO (Hazel/4) or LED (Hazel/Ariel) Light	270
Figure 43.	Hazel/Ariel Diagnostic Lights after Bring Up Test.	271
Figure 44.	AP/CSS Bringup Errors	273
Figure 45.	AP/DISK Bringup Errors	273
Figure 46.	AP/CSS ARIEL Bringup Errors	274
Figure 47.	AP/DIS ARIEL Bringup Errors	274
Figure 48.	Transmission Format	277
Figure 49.	Message Block Format	278
Figure 50.	Blast Reset State	280
Figure 51.	Blast Reset Sequence	281
Figure 52.	Protocol Example	282
Figure 53.	Protocol Example	283
Figure 54.	Protocol Example	283
Figure 55.	Protocol Example	284
Figure 56.	Nucleus Directory in Hazel/3	293
Figure 57.	Nucleus Directory in Hazel/4 without AP/CSS	294
Figure 58.	Nucleus Directory in Hazel/4 with AP/CSS	295
Figure 59.	Nucleus Directory in Hazel/Ariel	296
Figure 60.	Differences between MDCB and MDB data structures	314
Figure 61.	EDX Tasks Used for DIS I/O Frequency Comparison - 'NEW' format.	315
Figure 62.	EDX Tasks Used for DIS I/O Frequency Comparison - 'OLD' format.	316
Figure 63.	Comparison of DIS I/O Frequencies between Parallel Channel, SCA	317
Figure 64.	Comparison of DIS I/O Frequencies between Parallel Channel, SCA	318
Figure 65.	Comparison of DIS I/O Frequencies between Parallel Channel, SCA	318

IBM Internal Use Only

Figure 66. DSC TO S/I DUMP OVERVIEW	319
Figure 67. TRACES/TRACER Overrun	333

IBM Internal Use Only

1.0 INTRODUCTION

The EDX/J Language and Supervisor Description contains information relative to the function and operation of the Event Driven Executive for the Distributed System Controller (DSC). This manual is designed to assist the user in the design and implementation of application programs through the exclusive use of the command set described herein.

The various sections of this manual contain information on those topics with which a user must become familiar in order to:

- Use the command set to write programs.
- Use the supervisor utility functions which are provided in the system.

For a full understanding of this document, the reader must be familiar with:

- Distributed Sensor Subsystem (DSS) III, FORM NO. Z45-1081
- Distributed Sensor Subsystem (DSS) IV, FORM NO. E45-1289
- Distributed Interface System (DIS) architecture

(Ref. Distributed Interface System Summary FORM NO. Z45-0918)

For a full understanding of Hazel/Ariel DIS support, the reader must be familiar with:

- Remote Distributed Sensor Subsystem (RDIS) Functional Specification, FORM NO. E45-1353-0

The following publications are related to EDX/J:

- OS Assembler H General Information Manual , Order Number GC26-3758
- OS/VS and DOS/VS Assembler Language , Order Number GC33-4010
- OS Assembler Language , Order Number GC28-6514
- OS Assembler H Language , Order Number GC26-3771
- OS Assembler H Messages, Order Number SC26-3770
- Assembler H Version 2 Application Programming Guide , Order Number SC26-4036
- Assembler H Version 2 Language Reference , Order Number GC26-4037

Note: Either version (1 or 2) of the H assembler may be used.

IBM Internal Use Only

IBM Internal Use Only

2.0 SUMMARY OF CHANGES

The following is a brief summary of changes, enhancements, and fixes that have been made since the last revision.

2.1.1.1 Revision 7 - September 1986

This edition of the manual is a replacement for the previous level (Revision 5) dated June 1, 1984.

- Nucleus code for floating point comparisons was fixed. Use the FLOAT width, e.g. IF (A,LT,B,FLOAT), when comparing floating point numbers.
- Documented the fact that if you attempt to POST an ECB with 0 the 0 will be changed to -1 and -1 will be placed in the ECB.
- CONVTD was fixed to return the correct results when the field to be converted contains only EBCDIC zeros.
- The READTEXT macro does not support the SPACES, LINE, or COL parameters. The previous manual was incorrect.
- The PRINTTEXT macro does not support the COL parameter. The previous manual was incorrect.
- RESET was fixed so it does not wipe out pointers to tasks that are waiting on the ECB being reset.
- The nucleus (Hazel/Ariel only) now supports 256K words of memory (8 64K partitions).
- SELECT, TCBGET, and WHEREAS instructions were added to the language.
- Generalized ECB's (on INTBIT statements) are now posted when they are in partitions above 1. Previously, the system process checked.
- The manual was reorganized and updated in a number of places.
 - References to differences between EDX/J and EDX/J4 were removed. The current level of the language will run on both Hazel III's and Hazel IV's so there is no need to refer to the older version (previously called just EDX/J). The current level of the language (previously called EDX/J4) is documented here.
 - Commands are now listed in alphabetical order.
 - The manual was reorganized to be in (roughly) increasing order of difficulty. It starts with general EDX/J concepts, moves on to the language reference, to supervisor utilities, to the Hazel debug tool, etc.
 - Sections on the Hazel debug tool and EDX/J internals were added. The section on the debug tool was taken almost verbatim from Hazel Debug Tool: User Documentation by K. W. Hollingshead and Paul Pacholski.
 - The syntax summary was updated and page numbers were removed.
 - The table of contents was expanded. Hopefully, this will minimize the impact of removing the page numbers from the syntax summary.
 - Various tables were moved to the appendices.
 - The document was converted to GML and should be formatted with the starter set profile (DSMPROF3).
- Changes to EDX support for Hazel/Ariel were included. See "Appendix F. Hazel/4 - Hazel/Ariel DIS Support Compatibility" on page 311 for a summary of the changes to the EDX/J4 DIS commands for Hazel/Ariel.
 - New version of the following EDX/J4 commands were created for Hazel/Ariel: INTBIT, DISREAD, DISWRITE and LSA.
 - The following new Hazel/Ariel EDX/J4 commands were added: MDCB and DISIO.
 - The Internal Design Guide was updated to reflect changes in DIS support.

IBM Internal Use Only

- A description of the Hazel/Ariel Bringup Test was added to "IPL Bring Up Tests / Initialization" on page 270.
- A description of the Mini Debug (Hazel/3 and Hazel/4 with native display) was added ("HAZEL/3 and Hazel/4 - Mini Debug Overview" on page 177).
- A description of system traces was added ("TRACES" on page 209).
- The Hazel Debug Tool (Hazel/4 with AP/CSS) description was updated ("HAZEL with AP/CSS - Debug Tool Overview" on page 185).
- A description of the Disk Utility Program (DUT) was added ("Appendix J. DISK UTILITY PROGRAM (DUT)" on page 339).
- A description of the DSC to S/1 memory dump was added ("Appendix G. Memory Dump to S/1 Diskette" on page 319).
- A list of DSC and S/1 utility programs was added ("Appendix K. S/1 and DSC Utility Programs" on page 345).

3.0 SYSTEM OVERVIEW

3.1 GENERAL DESCRIPTION

The Event Driven Executive system is designed with the objective of being a highly responsive system supervisor controlled by a command set of instructions to facilitate the generation of data acquisition and control programs for event driven applications. Central to this objective is the support of multiple independent (time or event driven) applications, with minimum interaction.

It is helpful to consider two categories of requirements: those associated with real time control; and those associated with data reduction, storage and reporting.

Real-time control and data acquisition under the Event Driven Executive is supported by both a supervisor to manage the resources of the DSC and by application programs executing on the DSC. Their requirements include:

1. A command set for programming real-time application programs.
2. The ability to initiate any application program from a terminal or another application program, and to pass parameters to the new program.
3. Multitasking within each application program, with preemptive task switching.
4. Interval timing, with timing precision tolerances which may be deduced from the system specification for each mix of applications.
5. A relocating loader, so that an application program may use any available control storage area at the time of invocation.
6. The ability to operate independently of a host computer, except when an application program explicitly invokes interprocessor communication.

To simplify the preparation of application programs, symbolic addressing of memory locations and hardware device addresses can be used.

EDX/J provides the user with the ability to write application programs in a relatively short time and with little programming experience. The instructions provided by the EDX/J command set allow the user to control the DIS interface, operator terminal, timers, and host communications.

IBM Internal Use Only

3.2 SYSTEM COMPONENTS

The EDX/J system contains the following components:

- A Multi-programming / Multi-tasking Supervisor
- An Application Programming Language
- A Command Language Executor
- Operator Terminal (Keyboard and Display) support
- DIS Interface Support
- Host Communication Support

The supervisor provides control for a number of tasks simultaneously. The basic unit of work for the supervisor is a command. Commands are combined to form tasks. These tasks are assigned a priority by the user which is a measure of the relative criticality (usually time dependent) of its function.

The command library and the interpreter are provided for application program development. The EDX/J command set is designed to provide the most used functions while maintaining an elemental, easily learned structure. The commands are particularly designed for real time dynamic control of operation sequence, calculation, and decision making. Source programs may be written solely with these commands and assembled and formatted using host program preparation facilities.

The resulting loadable program consists of a table of constants which contain an average of three words for each command. The interpreter analyzes the commands and provides linkage to the appropriate system library routine to perform the command function.

In the multiprogramming environment, application programs are loaded into an sufficient area of contiguous control storage, when invoked by an operator command or the currently active task.

IBM Internal Use Only

3.3 SUMMARY OF FEATURES

- Multitasking/Multiprogramming Supervisor
- An Application Program Command Set
- 256 Task Execution Priorities
- Timing Precision = 1 millisecond
- Interactive Support for Operator Terminal
- Host Communication Support via either the native DIS interface (using the RS232C macrofunction) or the native host interface.
- Minimal Programming Experience Required

IBM Internal Use Only

3.4 MINIMUM PRACTICAL SYSTEM CONFIGURATION

1. For DSC Program Execution

- DSC with 64K words storage size
- Additional storage as defined by application program requirements
- Optional display with keyboard feature
- Selective macro function features

2. For Host Program Preparation

The minimum System/370 requirements for preparing Event Driven Executive based programs on the host computer are:

- An IBM System/370 with a partition or region of sufficient size to execute the program preparation facilities and the System/370 Utilities described in 'PROGRAMMING SYSTEMS' below.
- A 1052 Printer-Keyboard, or equivalent Operator's Console.
- A 1403 printer, or any device accepting SYSOUT=A data sets.
- Any IBM Direct Access Storage Device compatible with System/370.
- Any IBM Magnetic Tape Drive compatible with System/370.

IBM Internal Use Only

3.5 PROGRAMMING SYSTEMS

The IBM programs needed to support the Event Driven Executive for Host program preparation are:

- /370 ASM H (Version 1 or 2)
- JIB'/Toronto Macro Library - used under ASM/H for generating nucleus
- EDX/J Macro Library - used under ASM/H for generating application programs
- ASPEN post-processor - generates a more readable output listing from the ASM/H output listing
- OS/MVT 360S-CI-535 or OS/VS 5752-VS2 (MVS)
- Data Management 360S-CQ-513 or 5752-SC1-D0
- Utilities 360S-AS-506 or 5752-SC1-UA,U2,U3,U7, and U8

IBM Internal Use Only

IBM Internal Use Only

4.0 APPLICATION PROGRAMMING USER'S GUIDE

4.1 INTRODUCTION

This section is intended to be used as an application programming user's guide.

The general command format is discussed and examples of the key programming features are shown.

Finally, each command is described with detailed syntax rules, operand defaults, and examples of typical command usage - where there are multiple options. In some sections, short programming examples are included to aid the programmer's understanding of the commands.

The following paragraphs discuss the general layout and structure of programs written using the EDX/J command set.

In general, there are three basic EDX/J components involved in the execution of an application program:

- The EDX/J nucleus (operating system)
- The DIS Input/Output specifications
- The commands comprising the application program

One objective of this division is to minimize the dependence of the application program on a particular DSC configuration.

The chapter "Nucleus Generation" on page 215 describes the commands used to define the hardware features of the DSC. There are optional components in the Event Driven Execution nucleus which may or may not be included, depending upon the configuration of the particular DSC.

The DIS I/O definitions for an application program are established by use of the LSA or MDCB and INTBIT commands which are described in the sections "LSA - DSC" on page 107, "MDCB - ARIEL" on page 112 and "INTBIT - DSC" on page 95. These statements are normally grouped together and are most conveniently placed immediately after the PROGRAM statement near the beginning of a program. A user application program would have the following basic structure:

HAZEL/3, HAZEL/4 or HAZEL/ARIEL

PROGRAM

application program

LSA
LSA
INTBIT
ENDPROG
END

HAZEL/ARIEL only

PROGRAM

application program

MDCB
MDCB
INTBIT
ENDPROG
END

When several programs reference the same group of LSA, MDCB and INTBIT statements it may be convenient, at the user's discretion, to store these together as another member of the data set containing the system configuration statements. Using this technique, these statements are accessed by referring to this data set in the program preparation JCL.

IBM Internal Use Only

4.2 GENERAL COMMAND FORMAT

The EDX/J commands provide the user with the capability to design, develop, and implement application programs.

A source program starts with the PROGRAM statement and ends with ENDPORG and END statements. These are non-executable commands.

The basic unit of a program is a task. The PROGRAM statement defines a task which is referred to as the 'initial task' of the program. Many tasks may be active concurrently and asynchronously in a program. A task may be started or 'attached' from the initial task or from other tasks. Any combination of commands may be used within a task and it will be executed independently from other tasks. Tasks within a program may communicate with each other through common storage areas or through system commands and event control blocks. The facilities of the EDX/J supervisor provide the capability of synchronizing task execution. These functions are described in detail in the section "Task Control" on page 25.

Each command description will include a brief summary on the command operation, syntax, required parameters and the defaults used if parameters are omitted. Each operand (or parameter) is also listed and described.

EDX/J commands have the standard macro assembler format shown below:

```
label      op      par1,par2,...,parn,P1=P2=,...,Pn=
```

Syntactical coding rules are the same as Series/1 EDX. Some specific rules are restated here.

- All labels, instruction mnemonics, and parameter names must be alphanumeric strings 1 to 8 characters in length, the first being alphabetic.
- Statement labels must begin in column 1.
- To continue a statement on another line, code any character in column 72 and begin the next line in column 16. Examples shown in this manual may not conform to the column spacing conventions due to limitations in the length of printed lines.
- Several instructions allow the use of immediate data or constants. These are called self-defining terms and improve the flexibility and ease of programming in many cases.
- Labels that begin with '\$', '#', and '@' are reserved for system use.
- The parameters, par1,par2,...,parn, are labels, names, or values defined for each command. Parameter operands may also take the form NAME=name. This is called a 'keyword' operand.
- The parameter naming operands, P1=P2=,...,Pn=, are used to allow modification of command parameters at execution time. This is discussed further on the following pages.

Command formats are illustrated in the following example of a simple program with an initial task name of ADDTEN (Figure 1 on page 13).

IBM Internal Use Only

```

*MODLISTING
      COPY JIBPMAC
ADDTEN  PROGRAM  DOTEN
*****
* START OF PROGRAM DOTEN
*****
*SPACE 1
DOTEN   DO      10,TIMES      <=,
      ADD      COUNT,1      <=, INCREMENT COUNT 10 TIMES
      ENDDO
*SPACE 1
      PRINTTEXT 'RESULT ='
      PRINTNUM COUNT
      PROGSTOP
*EJECT
*****
* VARIABLES
*****
*SPACE 1
COUNT DATA      F'0'
*SPACE 1
      ENDPROG
*MODLISTING END
      END

```

Figure 1. A Simple EDX Program.

The statements '*MODLISTING' and '*MODLISTING END' are required for post-processing, see "Listing Control Instructions" on page 32. The statement 'COPY JIBPMAC' makes the system equates available to the program; it may not be required. The statements '*EJECT' and '*SPACE 1' control the appearance of the program assembly listing.

Note, that comment lines start with '*' in the first column, and that comments may be placed anywhere after the end of an EDX statement.

The first EDX/J4 statement (PROGRAM) is not executable. Its purpose is to define the program to the operating system. In the above example ADDTEN is the program name, and DOTEN is the label of the first statement that is executed after the program has been loaded.

The first command executed has the label DOTEN, which starts a DO loop. The next command adds 1 to a variable named COUNT, which was initialized to zero, by the DATA statement. The next command is ENDDO; it specifies the end of the DO loop. The ADD command is executed a total of ten times, and then a PRINTTEXT and PRINTNUM statement prints the result of this computation on a terminal. The PROGSTOP command terminates the program. The ENDPROG statement causes the compiler to drop the program's control blocks, in this case TCB. The END statement tells the compiler that there are no more EDX statements to be compiled. The ENDPROG and END statements must be the last two statements of an EDX/J application program.

IBM Internal Use Only

4.2.1 Address Indexing Feature

Two software registers (1 word each) are available to the user for each task and may be referenced in many instructions to provide 'indexed' addressing. The registers themselves are referenced by the names #1 and #2. In general, the registers may be used in the same manner as any other variable in the program. For example, the arithmetic (e.g. ADD), logical (e.g. AND), data movement (e.g. MOVE), and program sequencing (e.g. IF) commands may be used to set, modify, and test the registers. For example, the command

```
MOVE    #1,0
```

will set the register, #1, to the value 0. The command

```
MOVE    #2,A
```

will set #2 to the contents of the variable 'A'.

An example of the use of the register as the 'from' parameter

```
ADD     A,#1
```

Here, the contents of #1 will be added to the variable 'A'. It may be necessary to set a register to the address of a variable or vector. This is accomplished with the MOVEA command. For example,

```
MOVEA   #2,BUFFER
```

sets #2 to the address of the variable 'BUFFER'.

If an index register is used in the form

```
(parameter,#r)
```

then the effective address used is the sum of the contents of #r and 'parameter'. 'Parameter' can be either an address or a constant while 'r' specifies the index register to be referenced.

For example, if B = 2 then the indexed instruction

```
MOVE    A,(B,#1)
```

would be equivalent to the instruction

```
MOVE    A,(2,#1)
```

if #1 contained the address of 'B'. Both instructions would move into 'A', the content of the second word beyond 'B'.

The following example in Figure 2 illustrates the use of the indexing feature in a DO loop to find a value of -350 in a vector containing 1000 elements.

```
MOVE    #1,0
DO      1000,TIMES
  IF    ((BUF,#1),EQ,-350),GOTO,FOUND
  ADD   #1,1
ENDDO
      did not find a match
FOUND   MOVE    DISP,#1
```

Figure 2. Illustration of the Indexing Feature

The index register, #1, is set to zero, a DO loop is started of length 1000, thus, the buffer 'BUF' has a length of 1000 words (2000 bytes). A test is made on the first value of the buffer, and if there is a match, a branch to the label 'FOUND' is made. If not, the index register is incremented

IBM Internal Use Only

by 1 and the second value tested, etc. When the value -350 is found in the buffer, the displacement which is now contained in #1 is saved at the location 'DISP'.

For details of the use of these commands, the user is referred to the section "Data Movement, Logical and Arithmetic Functions" on page 18.

IBM Internal Use Only

4.2.2 Use of the Parameter Naming Operand Px=

In some programs it will be necessary to define instruction parameters at time of execution rather than during assembly. For example, the number of times a loop is to be executed may not be known at assembly time. The user may assign a name to a parameter by appending the mnemonic Px=name to the instruction. The parameter corresponding to the operand number 'x' (1,2,...) is given the name specified in Px=name. The following example may clarify a typical use of a Px= operand.

Program assembly of the two commands shown below produces the following tables of constants (generated DC instructions):

	MOVEA	M,NAME	
+	DC	A(\$MOVEA)	<- op code for MOVEA
+	DC	A(M)	<- addr of operand 1
+	DC	A(NAME)	<- addr of operand 2
	.		
	ADD	A,B,P1=M	
+	DC	A(\$ADD)	<- op code for ADD
+M	DC	A(A)	<- addr of operand 1
+	DC	A(B)	<- addr of operand 2
	.		

Execution of the MOVEA instruction changes the contents of the word originally shown as:

	+M	DC	A(A)
to:			
	+M	DC	A(NAME)

and execution of the ADD instruction would result in the addition of the contents of 'NAME' and 'B'.

IBM Internal Use Only

4.2.3 Symbolic I/O Assignments

The EDX/J commands DISWRITE, DISREAD, MDCB, LSA and WAIT refer to the DIS hardware via a 4 or 5 character name. For a macrofunction, the first three characters are LSA. For an interrupt bit, the first three characters are INT. The next two characters are the user identification, a number between 1 and 99. For example, if the user has two macrofunction locations and one interrupt bit assigned, then they are identified as LSA1, LSA2, and INT1 respectively. The assignment of the actual physical addresses is done prior to compilation of the application program using the LSA or MDCB and INTBIT commands. Therefore, all programs become independent of the physical location of the hardware. This prevents a program from using previously assigned addresses. The operating system will not load a program which uses the DIS interrupt bits assigned to a previously loaded program.

The details of making these assignments are described in the sections "LSA - DSC" on page 107, "MDCB - ARIEL" on page 112 and "INTBIT - DSC" on page 95.

4.2.4 Data Movement, Logical and Arithmetic Functions

Arithmetic, logical and data movement operations are performed with commands which are written according to a common general form. That form and the use of each parameter will be described here, while a specific format will be shown in the command summary to follow.

Syntax

label	OP	opnd1,opnd2,count,RESULT=,PREC=, P1=,P2=,P3=
Required		opnd1,opnd2
Defaults		count=1,RESULT=opnd1,PREC=S
Indexable		opnd1,opnd2,RESULT

Opcodes

The following operations are provided:

Data Movement:	MOVE	move data
	MOVEA	store address of data
Logical:	AND	logical 'and'
	IOR	logical 'or'
	EOR	logical 'exclusive or'
	SHIFTL	shift left, fill with zero
	SHIFTR	shift right, fill with zero
Arithmetic:	ADD	addition
	SUBTRACT	subtraction
	MULTIPLY	multiplication
	DIVIDE	division with remainder

Operands

opnd1
The name of the variable to which the operation applies. For example, the variables in SUBTRACT A,B correspond to the common algebraic notation A-B. If the RESULT= operand is not specified, then opnd1 is also the implicit result. This operand may not be a constant. (P1=)

opnd2
This operand determines the value by which the first operand is modified. Either the name of a variable or an explicit constant may be specified. (P2=)

Note: An explicit constant is limited to single precision (i.e. from -32768 to + 32767 decimal or from X'0000' to X'FFFF'). If an equated value is used as a immediate operand in any EDX/J instruction, it must be preceded by a plus sign. (ie. +#XYZ)

count
Specify the number of consecutive variables upon which the operation is to be performed. For example, ADD X,2,3 adds the constant 2 to the variables at word locations X, X+1, and X+2.

In general, the name specified for the first operand along with the count value, defines a vector to all of whose components the second operand is applied. Exceptions to this is the MOVE operation, which operates on corresponding components of two vectors.

For logical and data movement operations (but not arithmetic operations), the count operand is also used to specify the precision of the data. Since these operations are 'parallel', only one precision specification is required (i.e. the two operands and the result are implicitly of like precision). That specification may take one of the following forms:

IBM Internal Use Only

word precision - S or WORD
doubleword precision - D or DWORD

The precision specification may be substituted for the count specification, in which case the count defaults to 1, or it may accompany the count in the form of a sublist: (count,precision). For example, MOVE A,B,WORD and MOVE A,B,(1,WORD) are equivalent. (P3=)

Note:

1. This form of precision specification is also used for the PRINTNUM and GETVALUE commands described under "Terminal Input/Output" on page 28.
2. For operations with a large count value, a task switch may occur.

RESULT=

This operand may optionally be coded with the name of a variable or vector in which the result is to be placed. In this case, the variable specified by the first operand is not modified. RESULT= may not be specified for MOVE or MOVEA.

PREC=

This operand applies to arithmetic operations only, and when fully specified takes the form PREC=XYZ, where X applies to opnd1, Y to opnd2 and Z to the result. The value for each specification may be either S (single-precision) or D (double-precision), with restrictions which will be noted in the section "Mixed-precision Operations:". The full 3-operand specification may be abbreviated according to the following rules:

1. If no precision is specified, then all operands are single-precision.
2. If a single letter (S or D) is specified, then it applies to the first operand and result, with the second operand defaulted to single-precision.
3. If two letters are specified, then the first applies to the first operand and result, and the second to the second operand.

Data Representation:

Arithmetic operands are interpreted as signed binary integers with negative values represented in two's complement form. Single-precision (two byte) operands consist of 16 bits including sign, and double-precision (four byte) operands consist of 32 bits including sign. Logical operands are interpreted as bit strings of the appropriate length: word (16 bits) or doubleword (32 bits). single byte precision is not available.

Mixed-precision Operations:

Allowable precision combinations for arithmetic operations are listed in the following table:

opnd1	opnd2	result	abbreviation	remarks
S	S	S	S	default
S	S	D	SSD	RESULT must be specified
D	S	D	D	
D	D	D	DD	
D	S	S	DSS	DIVIDE only

Operations on Index Registers:

Index registers may generally be treated as ordinary single-precision arithmetic or logical variables. They are not, however, actually within

IBM Internal Use Only

the addressing range of the application program, and that a vector operation directed at the registers may not extend beyond #2.

Examples of Arithmetic Commands:

ADD	#1,2	Add 2 to index register 1.
SUB	A,B	Single-precision subtract.
MULT	C,D,RESULT=E,PREC=SSD	Double-precision product.
DIVIDE	VAL,(TAB,#1)	Second operand indexed.
SUB	A,(2,#2)	Subtr data at (2,#2) from A.
ADD	E,15,PREC=D	Add 15 to double-prec value.
MULT	C,A,RESULT=E,PREC=D	Single times double to double.
ALD	V1,A,16,RESULT=V2	Add A to 16 values of V1, result to V2.

Precision is specified with the PREC= operand.

Note:

1. All arithmetic overflows are indicated by placing X'8000' in the task code word referenced by the taskname (See PROGRAM, TASK). This word may be tested after any ADD, SUBTRACT, MULTIPLY, or DIVIDE command. For vector operations or operations where the count may be specified, the indicator (X'8000') is on if any one of the operations resulted in an overflow and in cases where more than one overflow may occur, the last 15 bits of the indicator specify one less than the number of overflows detected. For example, if the command:

ADD A,B,100

resulted in fifteen overflows, the task code word would be set to X'800E'.

2. The task code word is cleared before the execution of all arithmetic commands.

Examples of Logical Commands:

AND	A,X 00FF'	AND A with mask constant.
AND	B,A	AND B with A.
IOR	V,X,(50,WORD)	OR word X to 50 words at V.
EOR	C,D,1WORD	Doubleword exclusive OR.
SHIFTL	A,2	Shift A left 2.
SHIFTR	C,8,RESULT=E	Shift right 8, result to E.
SHIFTL	A,CT	Shift A left by value in CT.

Precision is specified with the count operand.

Examples of Data Movement Commands:

Precision is specified with the count operand and can be in the form of the sublist: (count,precision).

The use of either the FKEY or IKEY parameters or both allows the movement of data between the various partitions of memory:

MOVE	A,B	Move word, B to A.
MOVE	T,C' ',(64,1WORD)	Move EBCDIC blanks to 128 character field.
MOVE	V1,V2,16	Move V2 to V1, 16 words.
MOVE	SAVE,#1	Index register 1 to SAVE.
MOVE	#2,INDEX	Set index register 2 from INDEX.
MOVEA	PTR,A	Move address of A to PTR.
MOVE	D,C,(4,DWORD)	C to D, 4 doublewords.
MOVE	A,B,FKEY=1	Move B in part. 1 to A in current part.
MOVE	A,B,FKEY=L,IKEY=2	Move B in the part. defined in L to A in part. 2.

An indirect move from the address contained at X to location Y may be done as follows:

MOVE #1,X
MOVE Y,(0,#1)

An indirect move from the address contained at B to location A may be done as follows:

IBM Internal Use Only

```
MOVE #2,B,FKEY=1  
MOVE A,(0,#2),FKEY=LABEL,TKEY=2
```

If a word (ABC) is moved to the nth position past a label (SLABEL), a 2n value must be added to the label:

```
MOVE SLABEL+2N,ABC
```

If an equated value (#XYZ) is used as a immediate operand in any EDX/J instruction, it must be preceded by a plus sign:

```
ADD SLABEL,+#XYZ
```

IBM Internal Use Only

4.2.5 Derived Floating Point Functions

Derived Floating point functions share a common format.

The operands are interpreted as signed floating point numbers in standard precision. The second operand may be immediate data coded as integer value between -32768 and +32767. The immediate data will be converted to a floating point number prior to performing the operation.

Syntax

```

label    OP   opdn1, opnd2

Required      opdn1,opnd2
Defaults      None
Indexable     opdn1,opnd2

```

Operands

```

opdn1      The name of the variable where the result is to be stored.
            This must be an address.

opnd2      The quantity on which the operation is to be performed. This
            may be an address or immediate value.

```

Commands

Operation	Description
SQRT	Square root of opnd2, result in opdn1.
LOG	Common logarithm (base 10) of opnd2, result in opdn1.
LN	Natural logarithm (base e) of opnd2, result in opdn1.
EXP	Exponential of opnd2, result in opdn1.
PWR	Opnd 1 raised to the power in opnd2, result in opdn1.

After the operations of the derived floating point functions the return code is stored in the 'taskname'. See "PROGRAM" on page 131 and "TASK" on page 160.

Return Code	Description
-1	Successful
1	Operand too large, or overflow in operation
3	Divide-by-zero, or SQRT of negative value
5	Operand too small, or underflow in operation
7	Hardware check - the APU failed

Examples:

```

B      DATA  E'12.20'
A      DATA  E'0'

      SQRT  A,B
      PWR   A,4
      SQRT  (A,#1),2
      LN    A,(B,#2)
      EXP   (A,#1),(B,#2)

```

IBM Internal Use Only

4.2.6 Floating Point Arithmetic Instructions

Floating point arithmetic instructions share a common format which is described here. The specific format of each instruction will be shown in the command summary to follow.

Arithmetic operands are interpreted as signed floating point numbers in standard precision. Further, the second data operand may be immediate data coded as an integer value between -32768 and +32767. The immediate data will be converted to a floating point number prior to the arithmetic operation to be performed.

Syntax

label	operation	opnd1,opnd2,RESULT=, P1=,P2=,P3=
Required		opnd1,opnd2
Defaults		None
Indexable		opnd1,opnd2,RESULT

Operands

opnd1	The name of the variable to which the operation applies. For example, the variables in FSUB A,B correspond to the common algebraic notation A-B. If the RESULT= operand is not specified, then opnd1 is also the implicit result. This operand may not be a constant. (P1=)
opnd2	This operand determines the value by which the first operand is modified. Either the name of a variable or an explicit integer constant ('immediate data') between -32768 and +32767 may be specified. (P2=)
RESULT=	This operand may optionally be coded with the name of a variable in which the result is to be placed. In this case, the variable specified by the first operand is not modified. (P3=)

Floating Point EDX Commands

Command	Description
FADD	Signed addition of operand 1 and operand 2.
FSUB	Signed subtraction of operand 2 from operand 1.
FMULT	Signed multiplication of operand 1 by operand 2.
FDIVD	Signed division of operand 1 by operand 2.

Examples:

FADD	F1,F2,RESULT=F3
FSUB	L4,F2
FMULT	F1,F2
FADD	(0,#1),(2,#2),RESULT=ANSL
FSUB	L4,2

Operations on Index Registers

The index registers (#1 and #2) may not be used as operands in floating point operations since they are only 16 bits in length. The registers may, of course, be used to specify the address of a floating point operand.

Return Codes

IBM Internal Use Only

Return Code	Description
-----	-----
-1	Successful Completion
1	Floating Point Overflow
3	Floating Point Divide Check (Divide by 0)
5	Floating Point Underflow

The floating point operations return code is placed in the task code word (referred to by 'taskname', see PROGRAM,TASK). The codes are listed here and must be tested immediately after the floating point instruction is executed or the code may be destroyed by subsequent instructions.

IBM Internal Use Only

4.2.7 Task Control

A user written application program is composed of one or more tasks. The commands described in the following section are used to define tasks and to control which of the tasks are active at any given moment, plus other related functions.

Several programs, each composed of one or more tasks, may be loaded and run concurrently. When a user program task gains control of the system, a series of command operations is executed until a higher priority task becomes ready, at which time it will gain control of the system.

A program may have more than one independently operating task and these tasks may communicate with one another via common data storage locations or event control blocks.

It is the user's responsibility to write programs in such a way that the tasks operate in the desired sequence and terminate properly.

4.2.8 Program Sequencing

The commands 'IF', 'DO', 'SELECT' and 'GOTO' provide the means for sequencing a program through the correct path based on the data generated during execution. 'IF', 'SELECT' and 'DO' involve the use of relational statements which, based on a 'true' or 'false' condition, determine the next command to be executed. Relational statements consist of a combination of data elements and the relational condition mnemonics:

```
EQ ..... equal
NE ..... not equal
GT ..... greater than
LT ..... less than
GE ..... greater than or equal
LE ..... less than or equal
```

A relational statement has the general format:

```
(data1,relcond,data2,width)
```

where:

1. The 'width' parameter is optional. The default data width is one word (16 bits). The table below shows the allowed width specifications.

Specification	Data element width
WORD	1 word (16 bits)
DWORD	double word integer (32 bits)
FLOAT	floating point (32 bits)
n	n words (relcond may only be EQ or NE)

The last form (n) provides a means of comparing data strings. Note, the data element width in bytes is not available.

2. The 'relcond' parameter is one of the relational condition mnemonics,
3. The 'data1' and 'data2' are data elements, coded with the same syntax as EDX/J command operands. Only data2 can contain immediate data and only if width is not specified. (Immediate data is limited to a single precision value between -32768 to + 32767.)

Several forms of the 'IF', 'SELECT' and 'DO' commands are allowed as described in detail in the command description. The simplest form of the 'IF' command is

```
label IF (A,EQ,B)
```

If the word contained in the variable 'A' is equal to the word contained in the variable 'B', the next sequential command will be executed. This is called the true portion of the IF-ELSE-ENDIF structure. For example:

```
label IF (A,EQ,B)
:
: (code for true condition)
ELSE
:
: (code for false condition)
ENDIF
```

ELSE is an optional part of the structure and, if coded, the commands following it are referred to as the false part. Therefore, in the example above, the command following the ELSE command will be executed if 'A' is not equal to 'B'. If ELSE is not coded, control passes to the command following the ENDIF if the condition is false. The 'WHEN' part of the 'SELECT' command and the 'IF', 'SELECT' and 'DO' commands permit logically connected statements of the form:

```
statement,OR,statement
statement,AND,statement
```

The following examples are valid relational statements:

IBM Internal Use Only

Relational statement	Comments
(A,EQ,0)	A equal to 0, WORD
(A,NE,B)	A not equal to B, WORD
(A,LT,B,FLOAT)	A less than B, FLOAT
(DATA1,LT,DATA2,WORD)	DATA1 less than DATA2, WORD
(CHAR,EQ,C'AA',WORD)	CHAR equal to 'AA', WORD
(XVAL,GT,Y,DWORD)	XVAL greater than Y, DWORD
((A,#1),EQ,1)	(A,#1) equal to 1, WORD
((A1,#1),LE,(B1,#2))	(A1,#1) LE (B1,#2), WORD
(#1,EQ,1)	#1 equal to 1, WORD
(#1,GT,#2)	#1 greater than #2, WORD
((C,#2),EQ,CHAR,WORD)	(C,#2) equal to CHAR, WORD
(A,EQ,B,8)	A equal to B, 8 words
((BUF,#1),NE,DATA,N)	(BUF,#1) not equal to DATA, N words
(A,EQ,B),AND,(C,EQ,D)	A equal to B AND C equal to D, WORD
(#1,EQ,1),OR,((C,#1),EQ,D)	#1 equals 1 OR (C,#1) equals D, WORD

When writing code with structures, program readability is significantly enhanced by indenting nested structures. Two spaces for each nesting level is recommended. For example:

IF (A,EQ,B)	SELECT	SWITCH=A,(A,GT,-1)
DO	WHEN	(A,GE,Y)
WHILE,(X,NE,Y)	DO	10,TIMES
IF	ENDDO	
(#1,EQ,1)	WHEN	2
ENDIF	IF	(A,LT,#2)
ENDDO	IF	(#2,GT,B)
ELSE	ENDIF	
IF	WHEN	B
ENDIF	LEAVE	
	OTHER	
	ENDSELECT	

IBM Internal Use Only

4.2.9 Terminal Input/Output

EDX/J supports either a native display with keyboard or 3101 terminals.

The nucleus assumes that the keyboard is always present in the configuration, however the RAM/Video may or may not be. If it is not, all output messages are lost, the output operations are treated as 'no-ops' and a condition code of \$NOVIDEO is placed in the TCB code word to reflect this condition. Input operations must be completed. If, for example a QUESTION command is executed and no display is present, then the prompting message is lost. However, the task is suspended (and the keyboard is reserved) until the answer (Y or N) is entered.

EDX/J provides facilities to prevent conflicts among multiple users of the console. Each individual operation (read, write or control) acquires exclusive control of the console for its duration. However, if exclusive control is required for the duration of a sequence of commands, in order to print a report for example, then the ENQT and DEQT commands, described in this section, are used.

Exclusive access to a terminal is also obtained for the duration of a supervisor utility function initiated at the keyboard. For a description of those functions see "Supervisor Utility Functions" on page 173.

Error Handling:

I/O error logging and recovery procedures are generally carried out without a requirement for specific action by the application program. The error codes and system responses are to be defined later.

Data Representation - Output:

Normally, text (alphanumeric) data written to the console is represented internally as a string of EBCDIC characters, and the system translates to the code expected by the device.

Numeric data is represented internally as binary integers of single (two byte) or double (four byte) precision.

Data Representation - Input:

Programs generally request entry of text data in 'word' form, i.e., without imbedded blanks. When several words are entered on a line, they must be separated from each other and from any numeric entries on the same line by one or more blanks.

Numeric entries may be either decimal or hexadecimal, depending upon the program request. A leading '-' or '+' is considered to be part of a decimal entry. When multiple numeric entries are made on the same line, the entries may be separated by blanks or commas. In conjunction with this rule, there are two means of indicating omitted values in a numeric sequence, namely the use of an asterisk (*) or two consecutive commas. Omitted values result in no change to the corresponding internal values, and their interpretation therefore depends upon the utility or application program requesting the input. Allowable ranges for numeric input are as follows:

single-precision decimal	-32768 to 32767
double-precision decimal	-2147483648 to 2147483647
single-precision hexadecimal	0 to FFFF
double-precision hexadecimal	0 to FFFFFFFF

Note:

1. The RAM/Video display has a capacity of 1024 characters with 64 characters per line and 16 lines per display.
2. The 3101 terminal display has a capacity of 1920 characters with 80 characters per line and 24 lines per display.

Prompting and Advance Input:

IBM Internal Use Only

The dialogue between an application program or utility and the user is generally conducted through a series of 'prompting' messages requesting the required data. As the user becomes more familiar with the dialogue sequence, prompting becomes less necessary. The EDX/J commands, READTEXT and GETVALUE, include a 'conditional' prompting option which enables the user to enter data in advance, therefore inhibiting any associated prompting messages. By entering more than just the requested data on one line, subsequent input commands which have a PROMPT=COND keyword read data from the remainder of the (buffered) line, and issue a prompting message only when the line has been exhausted. If PROMPT=UNCOND (the default) is specified with an input command, the associated prompting message is issued and the system waits for the input data. The prompt message causes, as does every output message, cancellation of any outstanding advance input. Normally, prompting will result if no advanced input is present. However, under a unique situation, the user can avoid prompting even if there is no advanced input. The situation being; a PROMPT=COND keyword and NO prompt message present in either the GETVALUE or READTEXT command and NO advanced input. Under these conditions, the command acts like a no-op.

Attention Handling - Attention Key:

Program operation may be interrupted by pressing the keyboard ATTENTION key. On the native display the attention key is the PA1 key. On the 3101 terminal the attention key is the PF7 key.

When this key is recognized, the symbol '>' is displayed and the operator may enter either a system function code (see "Supervisor Utility Functions" on page 173), or a program function code defined by an active ATTNLIST (see "ATTNLIST" on page 37).

IBM Internal Use Only

4.2.10 Host Communication Functions

The OPEN, CLOSE, SEND and RECEIVE commands allow user programs to transfer messages through the Communications System. Syntax for the commands is the same on both the DSC and the Series/1 although the object code generated by the command macros may be different.

Each program using the Communication System must have one and only one OPEN command, which is executed before any of the other communication instructions in the program. After being OPENed the program may execute SEND and RECEIVE commands as desired, using multiple tasks if appropriate. Before the program is ended, a CLOSE command should be executed on the DSC and must be executed on the S/1. The DSC cancel support does an automatic close (if required) when canceling a program. The CLOSE command must be used instead of PROGSTOP by S/1 communication programs. Programs using the communication system MUST NOT be cancelled from the terminal.

A task in the DSC must not have the terminal ENQed (ENQT) at the time of a SLND, RECEIVE or LOADH.

The DEST statement is used to define a destination control block used by the SEND command.

The ENDTP statement marks the physical end of a program using the Communications System. It must follow all communication commands in the program and appear immediately before the ENDPROG statement.

IBM Internal Use Only

4.2.11 Disk Functions

The EDX/J READ and WRITE commands transfer data between memory and datasets resident on the DSC file System. The disk support on the DSC is fully compatible, physically, with that of the S/1. Diskettes can be interchanged freely between systems.

Some small differences do exist between the DSC and the S/1 disk support. These differences are in the area of the 'open' operation, in the syntax of the READ and WRITE commands and in the layout of the DSCB structure.

OPEN OPERATION The DSC support for 'opening a dataset' is built-in and is performed automatically (if required) as part of the execution of all READ and WRITE commands. No DSOPEN module is required as with the S/1.

READ/WRITE SYNTAX The READ and WRITE commands are fully compatible with the S/1 except for how cross-partition operations are specified. The READ / WRITE buffer need not be in the same partition as the READ or WRITE command. The DSC READ / WRITE commands provide a KEY= parameter to support cross-partition operations. The S/1 has no KEY= parameter and requires the user program to manipulate the TCB in order to specify a cross-partition operation.

DSCB LAYOUTS The size and layout of the DSCB's are different. Therefore, equates used on the S/1 are not compatible to equates needed by the DSC.

IBM Internal Use Only

4.2.12 Listing Control Instructions

Listing control instructions are used to format a program listing. By using these commands, the user can select the information to appear on the listing and the manner in which it is to be printed. With the exception of PRINT, listing control statements do not appear on the listing and no object code is generated for any of the commands.

The format used to describe these instructions is the same as that used for describing Event Driven Executive commands. However, they are part of the assembler facility and not the Event Driven Executive command set.

Normally, the output listing produced by ASM/H is automatically passed through the ASPEN post-processor in order to make the listing more legible. The source code must include both *MODLISTING and *MODLISTING END statements (both starting in column 1) to indicate where the post-processing is to be in effect. More than one *MODLISTING/*MODLISTING END pair may be used in order to select those portions of the listing to be post-processed.

Note: One *MODLISTING END statement must appear immediately before the END statement.

IBM Internal Use Only

4.3 COMMAND DESCRIPTIONS

Detailed descriptions of the Event Driven Executive commands are included in the following sections.

4.3.1 ADD

Function

Signed addition of opnd2 to opnd1.

Syntax

label	ADD	opnd1,opnd2,count,RESULT=,PREC= P1=,P2=,P3=
Required		opnd1,opnd2
Defaults		count=1,RESULT=opnd1,PREC=S
Indexable		opnd1,opnd2,RESULT

Operands

opnd1	The name of the variable to which the operation applies; it cannot be a constant. If the RESULT= operand is not specified, then opnd1 is also the implicit result. (P1=)
opnd2	This operand determines the value by which the first operand is modified. Either the name of a variable or an explicit constant may be specified. (P2=)
count	The number of consecutive variables upon which the operation is to be performed. (P3=)
RESULT=	This operand may optionally be coded with the name of a variable or vector in which the result is to be placed. In this case, the variable specified by the first operand is not modified.
PREC=	This operand takes the form PREC=XYZ, where X applies to opnd1, Y to opnd2, and Z to the result. The value for each specification may be either S (single-precision) or D (double-precision). See "Mixed-precision Operations:" on page 19 for restrictions and abbreviations of the PREC= operand.

See "Data Movement, Logical and Arithmetic Functions" on page 18 for a more detailed explanation and examples.

IBM Internal Use Only

4.3.2 ADDV

Function

Add two groups of numbers (vectors). Adds 'count' consecutive values in opnd2 to the corresponding value(s) in opnd1.

Syntax

label ADDV opnd1,opnd2,count,RESULT=,PREC=
 P1=,P2=,P3=

Required	opnd1,opnd2
Defaults	count=1,RESULT=opnd1,PREC=S
Indexable	opnd1,opnd2,RESULT

Operands

opnd1	The name of the variable to which the operation applies; it cannot be a constant. If the RESULT= operand is not specified, then opnd1 is also the implicit result. (P1=)
	Do not code software registers #1 or #2 for this operand. You can, however, use software registers to create and indexed address for opnd1.
opnd2	This operand determines the value by which the first operand is modified. Either the name of a variable or an explicit constant may be specified. (P2=)
count	The number of consecutive variables upon which the operation is to be performed. (P3=)
RESULT=	This operand may optionally be coded with the name of a variable or vector in which the result is to be placed. In this case, the variable specified by the first operand is not modified.
PREC=xyz	This operand takes the form PREC=xyz, where x applies to opnd1, y to opnd2, and z to the result. The value for each specification may be either S (single-precision) or D (double-precision). See "Mixed-precision Operations:" on page 19 for restrictions and abbreviations of the PREC= operand.

See "Data Movement, Logical and Arithmetic Functions" on page 18 for a more detailed explanation. Example:

```
ADDV V1,V2,32
      .
V1    DATA 32F'1'
V2    DATA 32F'2'
```

Adds each consecutive value in V1 to the corresponding value in V2. After execution V1 contains 32F'3'.

4.3.3 AND

Function

Logical 'and' of opnd2 to opnd1.

Syntax

label	AND	opnd1,opnd2,count,RESULT= P1=,P2=,P3=
Required		opnd1,opnd2
Defaults		count=1,RESULT=opnd1
Indexable		opnd1,opnd2,RESULT

Operands

opnd1	The name of the variable to which the operation applies; it cannot be a constant. If the RESULT= operand is not specified, then opnd1 is also the implicit result. (P1=)
opnd2	This operand determines the value by which the first operand is modified. Either the name of a variable or an explicit constant may be specified. (P2=)
count	The number of consecutive variables upon which the operation is to be performed. (P3=)
RESULT=	This operand may optionally be coded with the name of a variable or vector in which the result is to be placed. In this case, the variable specified by the first operand is not modified.

See "Data Movement, Logical and Arithmetic Functions" on page 18 for a more detailed explanation and examples.

IBM Internal Use Only

4.3.4 ATTACH

Function

ATTACH is used to attach or start a new task. If the named task is already active, no operation occurs.

Syntax

label	ATTACH	taskname,priority,CODE=,KEY=, P1=,P2=,P3=,P4=
Required		taskname
Defaults		CODE=-1,KEY=current partition
Indexable		None

Operands

taskname	Name of the task to be ATTACHed. This task must be defined with a TASK statement. (P1=)
priority	A priority to be assigned to the task. This priority will override and replace the one originally assigned in the TASK statement. It will remain in effect unless superceded by a subsequent ATTACH statement. (P2=)
CODE=	See the description of "TASK" on page 160 for a complete definition of "priority". A code word to be inserted in the first word of the control block of the task being attached. The code word may be referred to by the taskname. Sometimes when a task is attached from more than one point, it may be desirable to inform the task of the origin of the attachment. This code word value provides a simple mechanism for accomplishing this. Note: The code word should be examined immediately upon entry to the task since execution of certain commands (e.g. I/O) will cause it to be overlaid. (P3=)
KEY=	The key value specifies in which partition of memory the task to be attached resides. This operand can be a constant between 0 and 3 or the label of an address containing the key value. It cannot be indexed in any way. A value of 0 will not alter the presently active key value. An invalid key value is trapped during the instruction execution and will cause the machine to process check. (P4=)

IBM Internal Use Only

4.3.5 ATTNLIST

Function

ATTNLIST is used to provide entry to special user asynchronous attention interrupt handling routine. When the ATTENTION key (PA1 key on the native 3270 keyboard or the PF7 key on the 3101 terminal) is pressed, the system will query the user for a 1-8 character command. By convention, commands beginning with '\$' are reserved for system use. All other character combinations are allowed. If the command entered is specified in the list, control will be passed to the associated user routine. Such routines have the standard Event Driven Executive format. This provides the user with terminal I/O for interactive control of programs. These routines should be short because they are executed as a task of priority level 10, and may therefore interfere with the execution of any other user programs. They must end with the ENDATTN command.

Routines invoked by ATTNLIST commands operate as part of the System 'Keyboard Task' within the supervisor, rather than as part of the task in the user's program. Therefore the following instructions are NOT recommended to use in an ATTNLIST routine: DETACH, ENDTASK, ENQ, LOADH, PROGSTOP and WAIT.

Syntax

label ATTNLIST (cc1,loc1,...,ccn,locn)

Required	cc1,loc1
Defaults	None
Indexable	None

Operands

cc1, cc2, etc.

Any 1-8 alphanumeric characters, with the reservation that '\$' is reserved for system use as a first character. However, if the same command is specified more than once in the system (either in the same program, or in another program executing concurrently), only one entry is recognized and executed. Duplicate names must therefore be avoided.

loc1, loc2, etc.

Name of routine to be invoked.

Example of the ATTNLIST statement:

ATTNLIST (AB,PCODE1,CD,PCODE2)

PCODE1	MOVE CODE,1	Instructions executed by depressing
	ENDATTN	the ATTENTION KEY and entering 'AB'
PCODE2	POST EVENT,2	Instructions executed by depressing
	ENDATTN	the ATTENTION KEY and entering 'CD'

4.3.6 BUFFER

Function

BUFFER is used to define a data storage area with a standard format. The standard buffer contains an index and a length. The index may be used to indicate the current total number of words stored in the buffer. Both the index and the buffer are initialized to zero.

Certain commands, e.g. INTIME, have an optional indexing facility wherein they can be used to add new entries sequentially to a buffer by referencing and incrementing this index word. The index can be thought of as a subscript to a one dimensional array. If a buffer becomes full and is to be reused then the index word must be reset to zero. Examination of the index word would also indicate how many entries were currently in use in a buffer. The user may assign a name to the index word in the BUFFER statement to provide for such program references.

Syntax

```
label    BUFFER    count,INDEX=name
```

Required	count
Defaults	None
Indexable	None

Operands

label	A symbolic name to be assigned to the first word in the data portion of the buffer.
count	The length of the buffer in words (2 bytes = 1 word). ('count'+2 storage words will be allocated.)
INDEX =	INDEX=name specifies a symbolic name to be assigned to the buffer index word.

Example:

```
BUFF1    BUFFER 20,INDEX=INDX1
```

Generates:

	DATA	F'20'	.count word
INDX1	DATA	F'0'	.index word
BUFF1	DATA	20F'0'	.data area

IBM Internal Use Only

4.3.7 CALL

Function

CALL is used to execute a user written subroutine. Up to five parameters may be passed as arguments to the subroutine. Arguments must be single variables or constants, rather than a vector or buffer. The first command of the subroutine is identified by a SUBROUT statement.

Syntax

label CALL name,par1,...,par5,P1=,...,P6=

Required	name
Defaults	None
Indexable	None

Operands

name

The name of the subroutine to be executed.

par1,...

Explicit constants or symbolic references to constants which are to be passed to the subroutine. Updated values of these parameters are returned by the subroutine.

If the variable is enclosed in parenthesis, e.g. (par1), the address of the variable is passed to the subroutine. Any or all parameters can be bracketed individually.

Note:

1. The maximum number of subroutines that can be used in a program is 999.
2. Recursive subroutine CALLs are NOT supported in EDX/J.

IBM Internal Use Only

4.3.8 CANCEL

Function

CANCEL is used to delete a running program from the nucleus.

Syntax

```
label    CANCEL  loc,KEY=,P1=,P2=

Required      loc
Default       KEY=current partition
Indexable     None
```

Operands

loc
The address of a four word (eight characters) field containing the program name. The program name can be up to eight characters in length and must not start with a blank character. The unused space must be padded with blanks. (P1=)

KEY=
The key value specifies in which partition of memory the program to be cancelled resides. This operand can be a constant between 0 and 3 or an address containing the key value. It cannot be indexed in any way. A value of 0 will not alter the presently active key value. An invalid key value is trapped during the instruction execution and will cause the machine to process check. (P2=)

Note: A program cannot cancel itself by using the CANCEL command. If an attempt to do so is made or if the program to be cancelled is not found, then the TCB of the cancelling task will contain a X'FFFF' as a return code indicating a failed cancel. A successful cancel will have a return code of zero. If there is more than one program in the system under the same name, the program first loaded under that name will be cancelled.

4.3.9 CHKEY

Function

The partition key is saved in opnd2 and then set to a new value from opnd1.

Syntax

label	CHKEY	opnd1,opnd2
Required		opnd1
Defaults		none
Indexable		none

Operands

opnd1	The name of the variable which contains the new value of the partition constant. key.
	The key values must be between the values of 0 and 3 for Hazel/4 and between the values of 0 to 7 for Hazel/Ariel.
opnd2	This operand is used to save the partion key before it is set to opnd1.

Note:

1. If opnd2 is not specified the partition key will not be saved.
2. A key validation is done during execution of the instruction. If an invalid key is specified the current key will be saved in opnd2, but it will not be set to the value of opnd1.

IBM Internal Use Only

4.3.10 CLOSE

Function

The CLOSE command disconnects the program from the Communications System. Any data on the numbered receive ports is purged and new data arriving is rejected. Active SEND commands are aborted and the transfer may not be completed.

Syntax

label	CLOSE	STOP=,LOGMSG=
Required		None
Defaults		STOP= yes,LOGMSG= yes
Indexable		None

Operands

STOP=	Indicates whether a PROGSTOP is to be executed to terminate the program.
LOGMSG=	If specified YES then the message 'program name ENDED' is displayed on the terminal.

See also "OPEN" on page 121, "SEND" on page 148, and "RECEIVE" on page 141.

IBM Internal Use Only

4.3.11 CONVTB

Function

CONVTB is used to convert a binary value to an EBCDIC string. Both integer and floating point formats are provided. In addition, both the normal floating point notation, and E notation are provided.

Syntax

label CONVTB opnd1,opnd2,PREC=,FORMAT=,P1=,P2=

Required	opnd1,opnd2
Defaults	PREC=S,FORMAT=(6,0,I)
Indexable	opnd1,opnd2

Operands

opnd1 The name of an area in storage where the converted results will be placed. The address must be the leftmost byte of the area. The converted results will be in EBCDIC. (P1=)

opnd2 The name of the variable that is to be converted to EBCDIC. The user must know the format of the data. (P2=) The conversion support assumes the following:

Single Precision Integer	1 Word
Double Precision Integer	2 Words
Standard Precision Floating Point	2 Words

PREC= PREC= is used to specify the form of 'opnd2'. Defined values are shown below.

S	Single Precision Integer
D	Double Precision Integer
F	Standard Precision Floating Point

FORMAT=(w,d,t)

This parameter specifies the format of the output data (operand 1).

w Field width in bytes of EBCDIC field

d Number of digits to the right of decimal point Valid for floating point variables only. For integer values code a zero here.

t Type of EBCDIC Data

I	Integer XXXX
F	Real Number XXXX.XXX
E	Real Number of E Notation

Return Codes are passed through the 'taskname'. (See "PROGRAM" on page 131, "TASK" on page 160).

Return Code	Description
-----	-----
-1	successful
3	conversion error

The Convert Binary to EBCDIC instruction accepts both integer and floating point variables and converts them into an EBCDIC character string. The format of the EBCDIC character string is defined by the use of the PREC= and FORMAT= operands. The following examples will help to define the capabilities of this instruction. Integer Example:

IBM Internal Use Only

CONVTB TEXT,VALUE,PREC=S,FORMAT=(8,0,I)

VALUE	DATA	F'12345'
TEXT	TEXT	LENGTH=4

The value 12345 in the variable VALUE will be converted to EBCDIC at TEXT in the following format.

bbb12345

If conversion of double precision integers is required, then PREC=D is coded. Floating Point Example:

CONVTB	TEXT,VALUE,PREC=F,FORMAT=(16,4,F)
CONVTB	TEXT1,VALUE1,PREC=F,FORMAT=(20,14,E)

VALUE	DATA	E'62421.16'
VALUE1	DATA	E'4926139.2916'
TEXT	TEXT	LENGTH=8
TEXT1	TEXT	LENGTH=10

The following EBCDIC character strings would result:

TEXT=	bbbbbb62421.1600
TEXT1=	b.49261392916000Eb07

Note: The user is reminded that the conversion routines assume that the type of variable to be converted is as specified by the PREC= operand. If the internal format of the variable is something other than specified by the PREC= operand, incorrect results will occur.

IBM Internal Use Only

4.3.12 CONVTD

Function

CONVTD is used to convert an EBCDIC character string to a binary arithmetic value. Both integer and floating point variables are allowed.

Syntax

label CONVTD opnd1,opnd2,PREC=,FORMAT=,P1=,P2=

Required	opnd1,opnd2
Defaults	PREC=S,FORMAT=(6,0,I)
Indexable	opnd1,opnd2

Operands

opnd1

The name of a variable where the result of the conversion is to be stored. The user must ensure that enough space is reserved to accommodate the results. (P1=)

Single Precision Integer	1 Word (2 Bytes)
Double Precision Integer	2 Words (4 Bytes)
Standard Precision Floating Pt.	2 Words (4 Bytes)
Extended Precision Floating Pt.	4 Words (8 Bytes)

Note: The Extended Precision Floating Point is not supported by EDX/J.

opnd2

The address of the EBCDIC character string. It must be the address of the first character. See allowable range for data values under DATA in 'Data Definition' in this chapter. (P2=)

PREC=

This parameter (PREC) specifies the form of opnd1.

S	indicates Single Precision Integer
D	indicates Double Precision Integer
F	indicates Standard Precision Floating Point

FORMAT=(w,d,t)

This parameter specifies the format of the input data (operand 2).

w	Field width in bytes of EBCDIC field
d	Number of implied decimal positions if no decimal point is in input (valid for floating point only). For integer values code a zero.
t	Type of EBCDIC Data
I	Integer XXXXX
F	Real Number XXX.XX
E	Real Number in E Notation e.g. -3.765E6 (See "DATA" on page 47 for E notation format)

Return Codes are passed through the 'taskname'. (See "PROGRAM" on page 131, "TASK" on page 160).

Return Code	Description
-----	-----
-1	successful completion
1	no data in field
2	field omitted
3	conversion error

The Convert EBCDIC to Binary instruction accepts a variety of input formats. The following examples will help to define the various types accepted. Integer Example:

IBM Internal Use Only

CONVTD VALUE,TEXT,PREC=S,FORMAT=(8,0,I)

VALUE	DATA	F'0'
TEXT	TEXT	'12345',LENGTH=8

The value in EBCDIC, 12345, will be converted to a single precision binary value and stored at VALUE as X'3039'. Double precision integers can also be converted by using the PREC=D parameter and using a double word variable at VALUE. Floating Point Example:

CONVTD VALUE,TEXT,PREC=F,FORMAT=(10,2,F)
CONVTD VALUE1,TEXT1,PREC=F,FORMAT=(16,0,E)

VALUE	DATA	2F'0'
VALUE1	DATA	4F'0'
TEXT	TEXT	'100.5',LENGTH=5
TEXT1	TEXT	'0.1005E3',LENGTH=8

Both values shown in the TEXT statements would result in the same data being stored in the two DATA statements. The only difference would be that at VALUE1 an extended precision value would be stored.

The EBCDIC field should contain only those characters that are valid for the operation being performed:

Integers

Leading blanks

Digits 0 through 9

Trailing blanks

Floating Point

Leading blanks

Sign character + or -

Digits 0 through 9

Decimal Point

The character E, if E notation, followed by a sign character, + or -, or the digits 0 through 9.

If any of the following characters are found during the conversion, the following action will be taken:

Delimiters, or / 'End of Field' will be generated. If no data was found, a 'Field Omitted' return code will be returned.

All Blanks 'No Data in Field' return code will be returned.

Any Other Character 'End of Field' will be generated.

4.3.13 DATA**Function**

DATA is used to define one or more constants.

Syntax

label	DATA	expression
Required		expression
Defaults		None
Indexable		None

Operands

expression

A constant or an address specification consisting of 3 parts:

1. duplication factor (optional)
2. type (F, D, A, C, X, E, or L)
3. value

duplication factor

decimal constant describing number of times specification is to be repeated

type

F used to describe a decimal constant to be contained in one word (16 bits). Range = -32,768 to + 32,767.

A used to describe the address of a labelled location, to be contained in a word (16 bits).

C used to describe a character or character string to be stored in EBCDIC. Two characters are stored per word. If an odd number of characters is specified, a blank (X'40') is placed in the second character position of the last word.

X used to describe a hexadecimal constant. Four hexadecimal characters are stored per word. If the number of hexadecimal characters specified is not modulo four, the constant is padded to the left with one, two or three zeroes to complete the first word (i.e. the constant is right justified).

D used to describe a decimal constant to be contained in a double word (32 bits).

E standard precision floating point. (Two words.) Up to seven decimal digits allowed. Exponent range allowed is 10 to the -85 to 10 to the +75.

L extended precision floating point. (Four words.) Up to 16 decimal digits. Exponent range allowed is 10 to the -85 to 10 to the +75.

value for F, D, C, X, E and L, specify the constant within apostrophes; for A, specify the label within parentheses.

Exponent (E) notation is of the form SX.XXESYY where:

- S** optional sign char (+ or -) default is +
- X** characteristic 1 to 7 digits
- .** decimal point anywhere within characteristic
- E** designation of E notation
- Y** mantissa, range -85 to +75

Examples:

IBM Internal Use Only

DATA	F'1'	Decimal constant 1 (word).
DATA	128F'0'	128 words of zero.
DATA	D'400'	Decimal constant 400 (double word).
DATA	4D'-22'	4 decimal constants of minus 22 (double words).
DATA	A(LABEL)	The address of 'LABEL' (word).
DATA	C'XYZ'	Character string 'XYZ' (2 words).
DATA	80C' '	80 words - 2 blanks each.
DATA	5C'A'	5 words - each containing 'A'.
DATA	X'12'	Decimal constant 18 (word).
DATA	X'00123'	Decimal constant 294 (double word).
DATA	22X'A'	22 words containing decimal 10.
DATA	4E'1.2'	4 floating point values (2 words each value).
DATA	L'.123456E-40'	Extended prec. floating point in exponent form.

IBM Internal Use Only

4.3.14 DEQ

Function

DEQ is used to release exclusive control of a system or user resource. The user must always dequeue a resource previously reserved by the ENQ command. If the active task attempts to DEQ a resource not owned by the task then no action is taken and control is passed to the instruction following the DEQ.

Syntax

label	DEQ	resource,code,KEY=,P1=,P2=,P3=
Required		resource
Defaults		code=-1,KEY=current partition
Indexable		resource

Operands

resource	The symbolic name of the resource being DEQueued or released. (P1=)
code	A code word to be inserted into a queue control block (QCB) which defines the resource. The code word may be examined by referencing the symbolic name of the resource. This code may be used as a flag to indicate a condition or a status. (P2=)
KEY=	The key value specifies in which partition of memory the QCB that describes the resource being ENQed resides. This operand is described in the ATTACH instruction. (P3=)

IBM Internal Use Only

4.3.15 DEQT

Function

This command is used to release the console which was previously acquired with an ENQT command. Until this command is issued, the previous ENQT remains in effect and intervening ENQT commands from the same task are treated as no-ops.

Syntax

```
label      DEQT

Required   none
Defaults   none
Indexable  none
```

Examples of ENQT and DEQT:

```
ENQT
PRINTTEXT .....
DEQT
ENQT TERM1,BUSY=ALTERN
GETVALUE .....
DEQT
.
.
TERM1      IOCB  PAGESIZE=12,NHIST=3
ALTERN     WAIT  ECBX
.
.
```

Note:

1. The DEQT instruction is NOT to be used within an ATTNLIST routine.
2. If a DEQT command is issued and the terminal has been previously ENQTed, the contents of screen buffer (TBUF) are printed and the cursor is positioned at the beginning of the next line. In such case, if the 3101 display is used, a DEQT command will force the task issuing it to enter the wait queue for the duration of the data transfer. During this time, a task with a lower priority may become active.

IBM Internal Use Only

4.3.16 DEST

Function

The DEST statement creates a destination control block for use in a SEND command.

Syntax

label DEST name,port,NODE=,P2=

Required	name
Defaults	name= label on DEST statement,port=0,NODE=0
Indexable	None

Operands

name	The 6 character name assigned to the destination program.
port	port The receive port number in the destination program to which the message is directed. (P2=)
NODE=	The node number of the destination processor. (This is only meaningful on the SERIES/1)

IBM Internal Use Only

4.3.17 DETACH

Function

DETACH is used to remove a task from operational status.

If a task is re-ATTACHed after having issued a DETACH, execution will proceed with the next command after the DETACH in the re-ATTACHed task. (Therefore, if the task is to restart at the beginning, a GOTO statement should follow the DETACH statement.) An ENDTASK is required as the last statement in a block of code which begins with a TASK statement.

Syntax

label	DETACH	code,PI=
Required		None
Defaults		code = -1
Indexable		None

Operands

code	The post code to be inserted in the event (END ECB) named in the TASK statement of this task. (PI=)
------	---

Note: The DETACH instruction is NOT to be used within an ATTNLIST routine.

IBM Internal Use Only

4.3.18 DISIO - ARIEL

Function

See "DISREAD - DSC" on page 54, "DISWRITE - DSC" on page 57 and "Appendix F. Hazel/4 - Hazel/Ariel DIS Support Compatibility" on page 311.

Syntax

```
label      DISIO      mdcB,EVENT=,ERROR=,P1=,P2=
Required   mdcB
Indexable  none
```

Operands

mdcB
Specify the label of a HEADER MDCB (see "MDCB - ARIEL" on page 112). To execute the DISIO command, the programmer must define a HEADER MDCB followed by a MDCB body. (P1=)

EVENT=
Specify the label of an event control block (ECB) which will be posted with the completion code returned for the DISIO command. If the EVENT= parameter is specified, the task executing the DISIO command does not wait for the command to complete (except when COUNT=1 on MDCB), but continues to execute while the AP/DIS controller is executing the I/O command. In this case, the task is responsible to test or wait for the ECB to be posted with the I/O completion code, before it re-executes that same I/O instruction. If EVENT= is not specified, the task executing the DISIO command will be suspended until the DISIO command is completed. Note, that EVENT= and ERROR= are mutually exclusive.

ERROR=
Specify the label of an instruction to branch to if the command is unsuccessful.

DIS Channel Errors

If EVENT= is not specified, the taskname (first word of TCB) will contain a return code from a DIS I/O operation (See "TCBGET" on page 161). If EVENT= is specified, the ECB defined by the EVENT= keyword will contain the return code. These codes will be referenced by a label instead of an absolute number to allow future flexibility. The following is a list of all DIS return codes:

Label	RC	Description	LOCAL/REMOTE
----	--	-----	-----
\$OK	X'0000'	Command successful	BOTH
\$DISDXER	X'0001'	Hot return on data xfer	LOCAL BLOCK
\$DISSLER	X'0002'	Hot return on select	LOCAL BLOCK
\$cmderr	X'0008'	Dis Internal Command error	N/A
\$DISSLTO	X'0010'	Time out on select	BOTH
\$DISDXT0	X'0011'	Time out on data transfer	BOTH
\$DISRFC	X'0012'	Flag & CRC error on read	REMOTE BLOCK
\$DISLPAR	X'0016'	Parity error on read	LOCAL BLOCK
\$DISRCRC	X'0040'	CRC error on read	REMOTE BLOCK
\$DISRFLG	X'0050'	Flag error on read	REMOTE BLOCK
\$DIST	X'0090'	Hazel timed out waiting for AP (applicable only when COUNT=1)	BOTH

Example:

```
DISIO      READMDCB,ERROR=AIERR
:
AIERR      IF (taskname,EQ,$DISDXT0),GOTO,REDO
```

If a timeout condition occurred while executing the request, go to label REDO.

IBM Internal Use Only

4.3.19 DISREAD - DSC

Function

DISREAD is used to provide I/O operations on the macrofunction cards of the DIS interface.

The command is coded so that it is hardware address independent. The actual address accessed by the command is defined in the LSA statement.

Data is obtained from the DIS interface using the command tag specified, and placed in the user specified location. Note that the EDX/J supervisor is not intelligent with respect to the actual operation being performed on the macrofunction. It is the user's responsibility to ensure the proper sequence of commands is issued to the card in order to achieve the desired function.

Before issuing the CTC (Command Ta Code), a CTC=0 is issued if the macrofunction is not already selected. Then if the specified CTC is even, it is issued using the high-order byte of the word at loc as the data, then it is followed by another CTC with a value of 1 greater than that specified and using the low-order byte of the word at loc as the data. If the specified CTC is odd, it is issued using the low order byte of the word at loc as the data. If a CTC is specified, a CTC=0 is issued if required, followed by CTC=2 and CTC=3; if CTC is specified, a CTC=0 is issued if required, followed by CTC=5 only.

Syntax

label	DISREAD	LSAx, loc, CTC=, BITS=(u,v), COUNT=, ERROR=, P1=, P2=
-------	---------	---

Required LSAX,CTC=

Defaults COUNT=1,BITS=(u,v) specified in LSA command, loc (see description below). If Bits (U,V) is defaulted here, the system will use the Master U,V Bits which are in the MDB. If the MDB (U,V) bits are not specified, a full word or byte depending on the CIC will be transferred.

Indexable loc

Operands

LSAx Identifies the LSA command that defines the address of the macrofunction. (P1=)

loc Operand **loc** specifies the location where read data is to be placed. If **BITS=** is specified, the specified bits are right justified in the location, with higher order bits set to zero. If **CTC=** is odd, the byte appears in the low order byte of the word. (**P2=1**)

If loc is not specified, data is stored in or taken from the MDB (Macrofunction Data Block) associated with the device.

Specify a single digit hexadecimal value to be used as the command tag. The tag value must be in the range of 1 to F. If the value is even, a second command tag of value one greater will also be issued.

$$\text{BITS} = (u, v)$$

BITS= is used to specify the portion of bits read to be passed to the user. 'v' bits are selected starting at bit 'u'. The bits appear right justified with all higher order bits set zero.

If BITS= is not specified, the BITS= specification in the associated LSA statement, if specified, will be used as the default value.

IBM Internal Use Only

Note: If a single byte is read (CTC is odd), the byte will appear in the high order byte if BITS=(0,16) is also specified. The low order byte would be zero.

COUNT=

Specify the number of times the command tag sequence is to be issued. The data will be fetched from or stored into the next word location each time through the specified command sequence.

Important: Long data transfers on the DIS interface may lock out all other operations (including a task of higher priority) on the system for the duration of the transfer since the DIS interface has a higher priority than the EDX/J interpreter. Discretion should be exercised in the use of the COUNT parameter.

ERROR=

Specify the label of an instruction to branch to if the command is unsuccessful. If ERROR= is not coded, execution will proceed sequentially. In either case the task name will contain the completion code of the operation (See PROGRAM, TASK).

Examples:

DISREAD LSA1,CTC=3

A read operation using CTC=3. The byte read will appear at location LSA1 bits 8-15, bits 0-7=0.

DISREAD LSA20,DATA,CTC=2,COUNT=100

100 read operations using CTC=2 and CTC=3. The 100 words read will appear at DATA through DATA+99

DISREAD LSA21,DATA,CTC=2,BITS=(5,9)

A read operation using CTC=2 and CTC=3. Bits 5-13 of the word read will appear in bits 7-15 of the word at DATA. Bits 0-6=0.

DISREAD LSA21,DATA,CTC=9,BITS=(0,16)

A read operation using CTC=9. The byte read will appear in bits 0-7 of the word at DATA. Bits 8-15=0.

DISREAD LSA21,DATA,CTC=9,BITS=(5,6)

A read operation using CTC=9. Bits 5-7 of the byte read will appear in bits 10-12 of the word at DATA. Bits 13-15 and 0-9=0.

The taskname (first word of TCB) will contain a return code after a DIS I/O operation (See "TCBGET" on page 161). These codes will be referenced by a label instead of an absolute number to allow future flexibility. If any DIS I/O is used, these labels are automatically defined.

DIS Channel Errors:

Label	Code Value	Description
-----	-----	-----
\$OK	EQU X'0000'	Command successful
\$DISDXER	EQU X'0001'	Hot return tag on data transfer
\$DISSLER	EQU X'0002'	Hot return tag select sequence
\$DISMC	EQU X'0006'	Machine check on DISREAD
\$DISLTO	EQU X'0010'	TIMEOUT on select
\$DISDXT0	EQU X'0011'	TIMEOUT on data transfer

Console Command Errors:

Label	Code Value	Description
-----	-----	-----
\$SCRERR	EQU X'0020'	Screen error - line # contention
\$DCBOF	EQU X'0021'	DEC./BIN. overflow single precision
\$DCBDOF	EQU X'0022'	DEC./BIN. overflow double precision
\$HXB0F	EQU X'0023'	HEX./BIN. overflow single precision

IBM Internal Use Only

Example:

```
DISREAD  LSA1, DATA,CTC=4,ERROR=AIERR
:
:
AIERR    IF (taskname,EQ,+$DISDXT0),GOTO,REDO
```

If a timeout condition occurred while executing the request, go to label REDO.

IBM Internal Use Only

4.3.20 DISWRITE - DSC

Function

DISWRITE is used to provide I/O operations on the macrofunction cards of the DIS interface.

The command is coded so that it is hardware address independent. The actual address accessed by the command is defined in the LSA statement.

Data is obtained from the user specified location and placed on the DIS interface before issuing the command tag. Note that the EDX supervisor is not intelligent with respect to the actual operation being performed on the macrofunction. It is the user's responsibility to ensure the proper sequence of commands is issued to the card in order to achieve the desired function.

The DISWRITE command provides the capability of reading data from the macrofunction, modifying the data read and writing it back to the macrofunction, all within the same command.

Before issuing the CTC (Command Tag Code), a CTC=0 is issued if the macrofunction is not already selected. Then if the specified CTC is even, it is issued using the high-order byte of the word at loc as the data, then it is followed by another CTC with a value of 1 greater than that specified and using the low-order byte of the word at loc as the data. If the specified CTC is odd, it is issued using the low-order byte of the word at loc as the data. (e.g., if CTC=1 is specified, CTC=0 is issued if required, followed by CTC=2 and CTC=3; if CTC=5 is specified, a CTC=0 is issued if required, followed by CTC=5 only.)

Syntax

```
label    DISWRITE    LSAX,loc,CTC=,BITS=(u,v,rctc),COUNT=,ERROR=,
                                P1=,P2=
```

Required LSAX, CTC=

Defaults COUNT=1,BITS=(u,v) specified in LSA command, loc (see description below). If Bits (U,V) is defaulted here, the system will use the Master U,V Bits which are in the MDB. If the MDB (U,V) bits are not specified, a full word or byte depending on the CTC will be transferred.

Indexable loc

Operands

LSAx Identifies the LSA command that defines the address of the macrofunction. (P1=)

loc Operand loc specifies the location where the write data is to be obtained. If BITS= is specified, the specified number of bits are obtained from the low order bits of the word. If CTC= is odd, the byte is obtained from the low order byte of the word.

If loc is not specified, data is stored in or taken from the MDB (Macrofunction Data Block) associated with the device.

Specify a single digit hexadecimal value to be used as the command tag. The tag value must be in the range of 1 to F. If the value is even, a second command tag of value one greater will also be issued.

BITS=(u,v,rctc)

For DISWRITE, BITS= is used to specify the group of bits to be written. 'v' bits are written starting at bit 'u'. The 'v' low order bits of the word at 'loc' are used. The other bits outside this group are determined by issuing the CTC specified by 'rctc'. (i.e. The 'rctc' command is performed first. The data read is modified according to the 'u,v' parameters. Then it is written back out using the

IBM Internal Use Only

CTC=parameter.) If 'rctc' is even, a second command tag is issued to read a total of two bytes.

Note: Subparameter rctc is required if BITS= is used in DISWRITE.

If BITS= is not specified, the BITS= specification in the associated LSA statement, if specified, will be used as the default value.

COUNT=

Specify the number of times the command tag sequence is to be issued. The data will be fetched from or stored into the next word location each time through the specified command sequence.

Important: Long data transfers on the DIS interface may lock out all other operations (including a task of higher priority) on the system for the duration of the transfer since the DIS interface has a higher priority than EDX/J programs. Discretion should be exercised in the use of the COUNT parameter.

ERROR=

Specify the label of an instruction to branch to if the command is unsuccessful. If ERROR= is not coded, execution will proceed sequentially. In either case the first word of the task control block will contain the completion code of the operation (See PROGRAM,TASK).

Examples:

```
DISWRITE LSA5,CTC=5
```

A write operation using CTC=5. The byte written is obtained from the low order byte of the word at LSA5.

```
DISWRITE LSA5,DATA,CTC=4
```

A write operation using CTC=4 and CTC=5. The word written is obtained from location DATA.

```
DISWRITE LSA40,DATA,CTC=4,BITS=(5,9,2)
```

Initially, a read operation using CTC=2 and CTC=3 obtains a word from DIS. Bits 5-13 of the word read are made equal to bits 7-15 of the word at DATA. Bits 0-4 and 14-15 remain unchanged. The modified word is written using CTC=4 and CTC=5.

```
DISWRITE LSA40,DATA,CTC=7,BITS=(2,5,3)
```

Initially, a read operation using CTC=3 obtains a byte from DIS. Bits 2-6 of the byte are made equal to bits 11-15 of the word at DATA. Bits 0-1 and 7 remain unchanged. The modified byte is written using CTC=7. (If CTC=even in this case, the second byte written would be zero since no equivalent byte was read in.)

```
DISWRITE LSA40,DATA,CTC=7,BITS=(5,9,4)
```

Initially, a read operation using CTC=4 and CTC=5 obtain a word from DIS. Bits 5-13 of the word read are made equal to bits 7-15 of the word at DATA. However, since CTC=7 (odd), only the first byte of the word is written back out.

The taskname (first word of TCB) will contain a return code after a DIS I/O operation (See "TCBGET" on page 161). These codes will be referenced by a label instead of an absolute number to allow future flexibility. If any DIS I/O is used, these labels are automatically defined.

DIS Channel Errors:

IBM Internal Use Only

Label	Code Value	Description
-----	-----	-----
\$OK	EQU X'0000'	Command successful
\$DISDXER	EQU X'0001'	Hot return tag on data transfer
\$DISSLER	EQU X'0002'	Hot return tag select sequence
\$DISMC	EQU X'0006'	Machine check on DISREAD
\$DISSLTO	EQU X'0010'	TIMEOUT on select
\$DISDXT0	EQU X'0011'	TIMEOUT on data transfer

Console Command Errors:

Label	Code Value	Description
-----	-----	-----
\$SCRERR	EQU X'0020'	Screen error - line # contention
\$DCBOF	EQU X'0021'	DEC./BIN. overflow single precision
\$DCBDOF	EQU X'0022'	DEC./BIN. overflow double precision
\$HXB0F	EQU X'0023'	HEX./BIN. overflow single precision

Example:

```
DISWRITE LSA1,DATA,CTC=5,ERROR=AIERR
```

```
:
```

```
AIERR IF (taskname,EQ,+ $DISDXT0),GOTO,REDO
```

If a timeout condition occurred while executing the request, go to label REDO.

IBM Internal Use Only

4.3.21 DISREAD/DISWRITE - ARIEL

Function

See "DISREAD - DSC" on page 54, "DISWRITE - DSC" on page 57 and "Appendix F. Hazel/4 - Hazel/Ariel DIS Support Compatibility" on page 311.

Syntax

```
label    DISREAD  LSAX,loc,CTC=,BITS=(u,v),COUNT=,EVENT=,ERROR=,
              P1=P2,P5=P6=
label    DISWRITE LSAX,loc,CTC=,BITS=(u,v,rctc),COUNT=,EVENT=,ERROR=,
              P1=P2,P5=P6=
Required LSAX, CTC=, rctc is required if BITS= is used in DISWRITE.
Defaults COUNT=1, loc (see description below), If the (U,V) bits are
              defaulted here, a fullword or byte (depending on the CTC) or
              byte (depending on the CTC) will be transferred.
Indexable loc
```

Operands

LSAX
The label of a LSA structure which specifies the macrofunction card address on the I/O channel to be read or written. (P1=)

loc
A label of an appropriate sized buffer for reading or writing data. The size of the buffer must be compatible with the count specified in the command. For odd CTCs, the low order byte of each word in the buffer is sent or received. The DISREAD clears the upper byte when it stores a byte in the buffer. By not specifying a buffer, the system uses an internal byte/word field within the MDCB structure. The application program can access this transfer area in the MDCB by using the MDCB's P2= label. (P2=)

CTC=
Specify the single hexadecimal digit which defines the command tag. Range is 1 to F, however command tag 0 is also available for system level programming.

BITS=(u,v)
BITS=(u,v,rctc)
For DISREAD, BITS= is used to specify the portion of bits read to be passed to the user. 'v' bits are selected starting at bit 'u'. The bits appear right justified with all higher order bits set zero.
For DISWRITE, BITS= is used to specify the group of bits to be written. 'v' bits are written starting at bit 'u'. The 'v' low order bits of the word at 'loc' are used. The other bits outside this group are determined by issuing the CTC specified by 'rctc'. (i.e. The 'rctc' command is performed first. The data read is modified according to the 'u,v' parameters. Then it is written back out using the CTC=parameter.) If 'rctc' is even, a second command tag is issued to read a total of two bytes.

COUNT=
Specify the number of times the command tag sequence is to be issued. The data will be fetched from or stored into the next word location each time through the specified command sequence.
When COUNT=1 is specified all other operations (including tasks of higher priority) are locked out for the duration of the transfer. As a result, even if EVENT= is specified, the next sequential EDX instruction will not be executed until the DISIO instruction is completed. (P5=)

EVENT=
Specify the label of an event control block (ECB) which will be posted with the completion code returned for the I/O command. If the EVENT= parameter is specified, the task ex-

IBM Internal Use Only

executing the I/O command does not wait for the command to complete (except when COUNT=1), but continues to execute while the AP/DIS controller is executing the I/O command. In this case, the task is responsible to test or wait for the ECB to be posted with the I/O completion code, before it re-executes that same I/O instruction.

If EVENT= is not specified, the task executing the DISIO command will be suspended until the I/O command is completed.

Note that EVENT= and ERROR= are mutually exclusive.

ERROR= Specify the label of an instruction to branch to if the command is unsuccessful. (P6=)

ERROR= Specify the label of an instruction to branch to if the command is unsuccessful. If ERROR= is not coded, execution will proceed sequentially. In either case the first word of the task control block will contain the completion code of the operation (See PROGRAM,TASK).

DIS Channel Errors

If EVENT= is not specified, the taskname (first word of TCB) will contain a return code from a DIS I/O operation (See "TCBGET" on page 161). If EVENT= is specified, the ECB defined by the EVENT= keyword will contain the return code. These codes will be referenced by a label instead of an absolute number to allow future flexibility. The following is a list of all DIS return codes:

Label	RC	Description	LOCAL/REMOTE
-----	--	-----	-----
\$OK	X'0000'	Command successful	BOTH
\$DISDXER	X'0001'	Hot return on data xfer	LOCAL BLOCK
\$DISSLER	X'0002'	Hot return on select	LOCAL BLOCK
\$DISSLTO	X'0010'	Time out on select	BOTH
\$DISDXT0	X'0011'	Time out on data transfer	BOTH
\$DISRFC	X'0012'	Flag & CRC error on read	REMOTE BLOCK
\$DISLPAR	X'0016'	Parity error on read	LOCAL BLOCK
\$DISRCRC	X'0040'	CRC error on read	REMOTE BLOCK
\$DISRFLG	X'0050'	Flag error on read	REMOTE BLOCK
\$DIST	X'0090'	ARIEL timed out waiting for AP	BOTH
		(applicable only when COUNT=1)	

Example:

```
DISWRITE LSA1,DATA,CTC=5,ERROR=AIERR
:
:
AIERR IF (taskname,EQ,+$DISDXT0),GOTO,REDO
```

If a timeout condition occurred while executing the request, go to label REDO.

Examples:

See examples of DISREAD command in "DISREAD - DSC" on page 54 and DISWRITE command in "DISWRITE - DSC" on page 57.

4.3.22 DIVIDE

Function

Signed division of opnd1 by opnd2. The remainder is stored in the task code word and will be lost after the next DIVIDE or I/O operation. (For this reason, a DIVIDE command should not be placed between a STIMER command and the WAIT command associated with the STIMER.) Divide overflow is indicated by the special remainder x'8000' (see Note below). May be abbreviated DIV.

Warning: The remainder of a DIVIDE with PREC=DDD will be stored in the first two words of the TCB. Immediately after the operation, the remainder must be removed and the two TCB words cleared. Failure to do so may produce unpredictable results. Refrain from using the DIVIDE with PREC=DDD when PREC=DSS is adequate.

Syntax

label	DIVIDE	opnd1,opnd2,count,RESULT=,PREC= P1=,P2=,P3=
Required		opnd1,opnd2
Defaults		count=1,RESULT=opnd1,PREC=S
Indexable		opnd1,opnd2,RESULT

Operands

opnd1	The name of the variable to which the operation applies; it cannot be a constant. If the RESULT= operand is not specified, then opnd1 is also the implicit result. (P1=)
opnd2	This operand determines the value by which the first operand is modified. Either the name of a variable or an explicit constant may be specified. (P2=)
count	The number of consecutive variables upon which the operation is to be performed. (P3=)
RESULT=	This operand may optionally be coded with the name of a variable or vector in which the result is to be placed. In this case, the variable specified by the first operand is not modified.
PREC=	This operand takes the form PREC=XYZ, where X applies to opnd1, Y to opnd2, and Z to the result. The value for each specification may be either S (single-precision) or D (double-precision). See "Mixed-precision Operations:" on page 19 for restrictions and abbreviations of the PREC= operand.

See "Data Movement, Logical and Arithmetic Functions" on page 18 for a more detailed explanation and examples.

Note: When a task with a higher priority is waiting in the ready queue while a DIVIDE instruction with the count parameter greater than 255 is executing, a task switch will occur before the instruction is completed. The execution of the DIVIDE instruction will be resumed after the higher priority task terminates or enters a wait state.

IBM Internal Use Only

4.3.23 DO

Function

DO is used to initialize a loop. The DO loop must have an associated ENDDO command which defines the end of the loop. There are three forms of the DO command. 'DO UNTIL' and 'DO WHILE' provide a means of looping until or while a relational statement is true. The third form of the DO command causes a loop to be executed a specific number of times. In all of these forms a branch out of the loop is allowed.

Syntax

label	DO	count,TIMES,INDEX=,PI=
label	DO	UNTIL,statement
label	DO	WHILE,statement
Required		count or with UNTIL and WHILE, one relational statement
Defaults		None
Indexable		count or data1 and data2 in each statement

Operands

count	An explicit constant, or the label of a count, which defines the number of times the loop is to be executed. If count=0, then the loop will be executed one time.
TIMES	An optional operand which only serves to comment the command for program readability.
INDEX=	The label of a variable, defined by the user, which will be reset to zero before starting the DO loop and will be incremented by one each time the command immediately following the DO statement is executed. Therefore, the first time the loop is started the index will have a value of 1.
UNTIL	This parameter establishes a trailing decision loop, which is executed until the exit condition is true. Note that even if the condition is true initially, the loop will be executed one time.
WHILE	This parameter establishes a leading decision loop, which is executed as long as the exit condition is true. Note that if the condition is false initially, the loop will not be executed.
statement	A statement indicating the condition for the loop exit. This form is valid only following UNTIL or WHILE.

The description of the syntax and the examples of relational statements can be found in "Program Sequencing" on page 26.

Examples of DO and ENDDO

1. Simple DO

```
DO      100
:
:      (execute 100 times)
ENDDO
```

2. Simple DO with TIMES coded

```
DO      N,TIMES
:
:      (execute 'N' times)
ENDDO
```

IBM Internal Use Only

3. DO UNTIL

```
DO      UNTIL,(A,EQ,1000)
:
: (execute until A EQ 1000)
ENDDO
```

4. DO WHILE

```
DO      WHILE,(B,NE,C)
:
: (execute while B NE C)
ENDDO
```

5. Nested DO loops

```
DO      UNTIL,(A,EQ,B),OR,(*1,EQ,1000)
:
DO      10,TIMES
:
ENDDO
ENDDO
```

6. Nested DO loops and IF statements

```
DO      WHILE,(A,GT,B,DWORD)
IF      (CHAR,EQ,C'AA',WORD)
DO      40,TIMES
:
ENDDO
ELSE
:
ENDIF
ENDDO
```

4.3.24 DSCB**Function**

The DSCB statement creates a 'dataset control block' which is used by the READ and WRITE commands to access files on disk or diskette. The DSCB has two fields that is formatted by the user: the dataset NAME field, and the dataset VOLUME NAME field. The only other field in the DSCB is the EVENT field which contains an ECB. The ECB is located at the start of the DSCB and is addressable through the label given to the DSCB. The application program may test or wait on this ECB.

Syntax

label	DSCB	dsname,volname,P1=,P2=
Required		none
Defaults		none
Indexable		none

Operands

dsname	The 8 character name assigned to the dataset on file. The dsname need does not have to be provided in the source code, but the name must be inserted into to DSCB name field (using P1=) during run time and prior to the use of the DSCB in a READ or WRITE command.
volname	The 6 character name assigned to the volume containing the dataset to be accessed. The volume name or ID is an optional parameter for both source and run time environments. If the volume field is not to be used it must be set with blanks. The volname field is set with blank if not define in the source. (P2=)
Px=	The P1= parameter allows the application program to enter or change a dataset name in a DSCB during run time. The P2= parameter allows the application program to enter or change the volname in a DSCB during run time.

Disk Return Codes

0	Good Return Code
1	File Not Ready
2 *	Seek Error
3 *	Read / Write Error
4	Formatting Error
5	CTS Error
6	Timeout Error (hardfile)
12	Empty File Error
13	Dataset Not Found Error
15	End of File

* - The disk support retries 3 times for Seek errors and 10 times for Read/Write errors before returning the error condition.

Note: DSCB is NOT an executable statement and should therefore be placed in a data area of a program, not in the middle of executable commands.

IBM Internal Use Only

4.3.25 ECB

Function

ECB generates an Event Control Block (ECB) which is the symbolic representation of an event.

Normally, this statement will not be needed for writing application programs. Event Control Blocks are automatically generated for the user as a consequence of naming an 'event' in other commands. However, it may be used for special purposes such as controlling their location within a program.

Note: ECB is not an executable statement and should therefore be placed in a data area of a program, not in the middle of executable commands.

Syntax

label	ECB	code
Required	label	
Defaults	code = -1	
Indexable	None	

Operands

code	Initial value of the code field (word 1). If this word is non-zero when a WAIT is issued, no wait will occur.
------	---

Note:

1. ECB's can NOT be coded as consecutive DATA statements, for example:
DATA 3F'0'
2. ECB's can NOT be moved (for instance, into SYSCOM) with the MOVE statement. See "SYSCDEF" on page 157 and "SYSCOPEN" on page 159 for techniques for moving ECB's into SYSCOM.

IBM Internal Use Only

4.3.26 EJECT

Function

The EJECT statement causes the next line of the listing to appear at the top of a new page, therefore providing a convenient way to separate sections of a program. There is no effect to the title, if one has been defined.

Syntax

EJECT

Operands

None

Note: The EJECT command has no effect while the post-processor is active. To perform the same function during post-processing, the *EJECT statement (starting in column 1) is used. See "Listing Control Instructions" on page 32 for more information on the post-processor.

IBM Internal Use Only

4.3.27 ELSE

Function

ELSE defines the start of the "false" code associated with the preceding IF statement. The end of the false code is the next ENDF statement.

Syntax

label	ELSE
Required	None
Defaults	None
Indexable	None

Operands

None

See "IF" on page 93 for examples.

IBM Internal Use Only

4.3.28 END

Function

END must be coded as the last statement in a user's program.

Syntax

END

Required
Defaults
Indexable

Must be coded as shown above.
None
None

Operands

None

IBM Internal Use Only

4.3.29 ENDATTN

Function

ENDATTN is used to end an asynchronous attention interrupt processing routine, as described under ATTNLIST, and is the last statement of that routine.

An attention interrupt handler should be a very short routine that is used to provide operator terminal initiation or control of application routines.

Syntax

label	ENDATTN
Required	None
Defaults	None
Indexable	None

Operands

None

See "ATTNLIST" on page 37 for examples.

4.3.30 ENDDO

Function

ENDDO is used to define the end of a DO loop. It must be preceded by a DO command. Twenty nested loops are allowed, and each must be defined by a DO and an ENDDO.

Syntax

label ENDDO

Required	None
Defaults	None
Indexable	None

Operands

None

See "DO" on page 63 for examples.

IBM Internal Use Only

4.3.31 ENDIF

Function

ENDIF indicates the end of an IF-ELSE structure. If ELSE is coded, ENDIF indicates the end of the false code associated with the preceding IF statement. If ELSE was not coded, ENDIF indicates the end of the true code associated with the preceding IF statement.

Syntax

label	ENDIF
Required	None
Defaults	None
Indexable	None

Operands

None

See "IF" on page 93 for examples.

IBM Internal Use Only

4.3.32 ENDPROG

Function

ENDPROG must be coded as the next to the last statement in a user's program.

Syntax

ENDPROG

Required
Defaults
Indexable

Must be coded as shown above.
None
None

Operands

None

IBM Internal Use Only

4.3.33 ENDSELECT

Function

ENDSELECT indicates the end of the associated SELECT block.

Syntax

label ENDSELECT

Required	None
Defaults	None
Indexable	None

Operands

None

See "SELECT" on page 145 for examples.

IBM Internal Use Only

4.3.34 ENDTASK

Function

ENDTASK is used to define the end of a block of code associated with a specific task (defined with a preceding TASK statement). Each task requires one ENDTASK as its final statement.

Syntax

ENDTASK code,P1=

Required	None
Defaults	code=-1
Indexable	None

Operands

code	The post code to be inserted in the event (END ECB) named in the TASK statement of this task. (P1=)
------	---

Note: The ENDTASK instruction is NOT to be used within an ATTNLIST routine.

IBM Internal Use Only

4.3.35 ENDTP

Function

The ENDTP statement creates a chain pointer referenced in the Open Control Block and used by the CLOSE command. It must appear in any program using the OPEN command and must follow all SEND commands. It is recommended that the ENDTP be placed immediately before the ENDPROG statement.

Syntax

label	ENDTP
Required	None
Defaults	None
Indexable	None

IBM Internal Use Only

4.3.36 ENQ

Function

ENQ is used to acquire exclusive control of a system or user resource.

A resource is a logical or physical entity (e.g. an I/O device or subroutine) which must be used in a serial fashion. Queuing is the process of acquiring exclusive control in order to ensure serial, one at a time, use. In general, there are two types of resources, system and user. System resources are those which may be shared serially by all user programs and are defined by symbolic names which are known broadly across the system. User resources are shared serially by different parts of one user program and are identified by symbolic names known only within that user program. A resource, system or user, is described symbolically by an Queue Control Block (QCB).

Syntax

label	ENQ	resource,BUSY=,KEY=,P1=,P2=,P3=
Required		resource
Defaults		KEY=current partition
Indexable		resource

Operands

resource	The symbolic name of the resource to be queued for. (P1=)
BUSY=	The address of the command to receive control if the requested resource is not available. If the resource is busy and this operand is not specified, the requesting task will be placed in a wait state until it is available. (P2=)
KEY=	The key value indicates which memory partition contains the QCB being ENQed. This operand is described in the ATTACH instruction. (P3=)

Examples:

```
ENQ QCB
ENQ QCB,BUSY=A,KEY=$KEY
```

IBM Internal Use Only

4.3.37 ENQT

Function

The command ENQT is used to acquire exclusive control of the console until the DEQT command is executed. This command is also used to establish terminal configuration parameters which will be in effect during the period of exclusive access.

Syntax

label	ENQT	name,BUSY=
Required		None
Defaults		name (see below)
Indexable		None

Operands

name

This parameter is used to specify the IOCB statement which defines the configuration in which the console is to be used. If this parameter is omitted, the console is used in its normal configuration as specified in the TERMINAL command of the System Generation.

BUSY=

The console may be under the control of another task or a supervisor utility function at the time of the request. The requesting task is then placed in a queue, waiting for the device, and its operation is suspended until all other users preceding it have been serviced. The BUSY= operand allows the task to detect such a 'busy' condition before it is placed in the queue. Code BUSY= with the label of the command at which execution is to proceed if the terminal is in use.

See "DEQT" on page 50 for examples.

Note: If ENQT command with IOCB specified is issued, the contents of screen buffer (TBUF) are printed and the cursor is positioned at the beginning of the next line. In such case (only if 3101 display is used) an ENQT command will force the task issuing it to enter the wait queue for the duration of the data transfer. During this time, a task with a lower priority may become active.

IBM Internal Use Only

4.3.38 EOR

Function

Logical 'exclusive or' of opnd2 to opnd1.

Syntax

label operation opnd1,opnd2,count,RESULT=
 P1=P2=P3=

Required	opnd1,opnd2
Defaults	count=1,RESULT=opnd1
Indexable	opnd1,opnd2,RESULT

Operands

opnd1	The name of the variable to which the operation applies; it cannot be a constant. If the RESULT= operand is not specified, then opnd1 is also the implicit result. (P1=)
opnd2	This operand determines the value by which the first operand is modified. Either the name of a variable or an explicit constant may be specified. (P2=)
count	The number of consecutive variables upon which the operation is to be performed. (P3=)
RESULT=	This operand may optionally be coded with the name of a variable or vector in which the result is to be placed. In this case, the variable specified by the first operand is not modified.

See "Data Movement, Logical and Arithmetic Functions" on page 18 for a more detailed explanation and examples.

IBM Internal Use Only

4.3.39 EQU

Function

Assigns a value to a label. The value is a word in length. You can use labels you define with the EQU statement in other instructions that permit the use of labels. The 'value' may be an integer constant, another label, an expression containing arithmetic operators, or an asterisk.

Syntax

label	EQU	value
Required		label,value
Defaults		none
Indexable		none

Operands

label	The label to be assigned a value. Label names must be unique within a program.
-------	--

value	An integer constant, another label, an expression containing an arithmetic operator, or an asterisk. The asterisk points to the next available word in storage. It allows you to generate convenient labels that you can use within your program.
-------	---

If value is coded as a label, it must be defined when the system process the EQU statement. For example, if you code:

A EQU B

you must have previously defined the label B in your program.

Examples:

	MOVE	D,A	move contents at address x'0002' to D
	MOVE	D,+A	move 2 to D
A	EQU	2	assign the value of 2 to A
B	EQU	A	assign the value of A (in this case 2) to B
C	EQU	B+2	assign the value of B+2 to C

Notes:

1. When you use the label on the EQU statement as an operand in another instruction, the system interprets the label as a storage address unless you include a plus sign (+) before it. The system interprets a label preceded by a plus sign as a constant.
2. If you equate a DATA statement with a label, the system interprets the label as the address of the DATA statement. If you attempt to use this label with a plus sign, however, the label will no longer point to the DATA when the load point of the program changes.

IBM Internal Use Only

4.3.40 EXP

Function

Exponential of opnd2. The result is placed in opnd1.

Syntax

label	EXP	opnd1,opnd2,P1=,P2=
Required		opnd1,opnd2
Defaults		none
Indexable		opnd1,opnd2

Operands

opnd1	
opnd2	The name of the variable where the result is stored.

The quantity on which to operation is to be performed. May be an address or an immediate value (constant). If opnd2 is an address then the data at that address is assumed to be a signed floating point number of standard precision (double word). If opnd2 is immediate data it may be coded as an integer value between -32768 and 32767 and will be converted to a floating point number prior to performing the operation.

See "Derived Floating Point Functions" on page 22 for a more detailed explanation and examples.

After the operation is performed the return code is stored in the first word of the task control block (see "TCBGET" on page 161). Valid return codes include:

Return Code	Description
-----	-----
-1	successful
1	Operand too large, or overflow in operation
3	Divide-by-zero
5	Operand too small, or underflow in operation
7	Hardware check. The APU failed to complete the execution of the command.

IBM Internal Use Only

4.3.41 FADD

Function

Signed floating point addition of opnd1 to opnd2.

Syntax

label	FADD	opnd1,opnd2,RESULT=, P1=,P2=,P3=
Required		opnd1,opnd2
Defaults		none
Indexable		opnd1,opnd2,RESULT

Operands

opnd1	The name of the data area to which opnd2 is added. Opnd1 cannot be a self-defining term (constant or equate). The result of the operation is stored in opnd1 unless RESULT is coded. (P1=)
opnd2	The value to be added to opnd1. May be a self-defining term (constant or equate) or the label of a data area. If opnd2 is the address of a data area then the data at that address is assumed to be a signed floating point number of standard precision (double word). If opnd2 is immediate data it may be coded as an integer value between -32768 and 32767 and will be converted to a floating point number prior to performing the operation. (P2=)
RESULT=	The name of a data area where the result is to be placed. If not specified, the result is placed in opnd1. (P3=)

See "Floating Point Arithmetic Instructions" on page 23 for a more detailed explanation and examples.

After the operation is performed the return code is stored in the first word of the task control block (see "TCBGET" on page 161). Valid return codes include:

Return Code	Description
-----	-----
-1	successful
1	floating point overflow
5	floating point underflow
7	Hardware check. The APU failed to complete the execution of the command.

Note: The software registers (#1 and #2) may **not** be used as operands in floating point operations since they are only 16 bits in length. The registers may, of course, be used to specify the address of operands.

IBM Internal Use Only

4.3.42 FDIVD

Function

Signed floating point division of opnd1 by opnd2 (opnd1/opnd2).

Syntax

label	FDIVD	opnd1,opnd2,RESULT=, P1=,P2=,P3=
Required		opnd1,opnd2
Defaults		none
Indexable		opnd1,opnd2,RESULT

Operands

opnd1	The name of the data area by which opnd2 is divided. Opnd1 cannot be a self-defining term (constant or equate). The result of the operation is stored in opnd1 unless RESULT is coded. (P1=)
opnd2	The value to divide opnd1 by. May be a self-defining term (constant or equate) or the label of a data area. If opnd2 is the address of a data area then the data at that address is assumed to be a signed floating point number of standard precision (double word). If opnd2 is immediate data it may be coded as an integer value between -32768 and 32767 and will be converted to a floating point number prior to performing the operation. (P2=)
RESULT=	The name of a data area where the result is to be placed. If not specified, the result is placed in opnd1. (P3=)

See "Floating Point Arithmetic Instructions" on page 23 for a more detailed explanation and examples.

After the operation is performed the return code is stored in the first word of the task control block (see "TCBGET" on page 161). Valid return codes include:

Return Code	Description
-----	-----
-1	successful
1	floating point overflow
3	divide by zero
5	floating point underflow
7	Hardware check. The APU failed to complete the execution of the command.

Note:

1. The software registers (#1 and #2) may **not** be used as operands in floating point operations since they are only 16 bits in length. The registers may, of course, be used to specify the address of operands.
2. When a task with a higher priority is waiting in the ready queue while a FDIVD instruction with the count parameter greater than 255 is executing, a task switch will occur before the instruction is completed. The execution of the FDIVD instruction will be resumed after the higher priority task terminates or enters a wait state.

IBM Internal Use Only

4.3.43 FMULT

Function

Signed floating point multiplication of opnd1 by opnd2.

Syntax

```
label    FMULT      opnd1,opnd2,RESULT=,
                    F1=,P2=,P3=

Required      opnd1,opnd2
Defaults      none
Indexable     opnd1,opnd2,RESULT
```

Operands

opnd1
The name of the data area by which opnd2 is multiplied. Opnd1 cannot be a self-defining term (constant or equate). The result of the operation is stored in opnd1 unless RESULT is coded. (P1=)

opnd2
The value to multiply opnd1 by. May be a self-defining term (constant or equate) or the label of a data area. If opnd2 is the address of a data area then the data at that address is assumed to be a signed floating point number of standard precision (double word). If opnd2 is immediate data it may be coded as an integer value between -32768 and 32767 and will be converted to a floating point number prior to performing the operation. (P2=)

RESULT=
The name of a data area where the result is to be placed. If not specified, the result is placed in opnd1. (P3=)

See "Floating Point Arithmetic Instructions" on page 23 for a more detailed explanation and examples.

After the operation is performed the return code is stored in the first word of the task control block (see "TCBGET" on page 161). Valid return codes include:

Return Code	Description
-----	-----
-1	successful
1	floating point overflow
5	floating point underflow
7	Hardware check. The APU failed to complete the execution of the command.

Note:

1. The software registers (#1 and #2) may **not** be used as operands in floating point operations since they are only 16 bits in length. The registers may, of course, be used to specify the address of operands.
2. When a task with a higher priority is waiting in the ready queue while a FMULT instruction with the count parameter greater than 255 is executing, a task switch will occur before the instruction is completed. The execution of the FMULT instruction will be resumed after the higher priority task terminates or enters a wait state.

IBM Internal Use Only

4.3.44 FPCONV

Function

FPCONV is used to convert integer values to or from floating point numbers.

Syntax

label FPCONV opnd1,opnd2,COUNT=,PREC=,
 P1=,P2=,P3=

Required	opnd1,opnd2
Defaults	COUNT=1,PREC=FS
Indexable	opnd1,opnd2

Operands

opnd1	The address (label or index register reference) to receive the output of the conversion. (P1=)
opnd2	The address of the data input to the conversion. 'opnd2' may also be immediate data in the form of an integer constant between -32768 and +32767. (P2=)
COUNT=	The number of values, beginning at 'opnd2', to be converted and stored at locations beginning at 'opnd1'. If 'opnd2' is immediate data, it will be converted and stored beginning at the location defined by 'opnd1' for as many locations as are defined by the 'COUNT=' operand. (P3=)
PREC=	Defines the type and precision of 'opnd1' and 'opnd2'. Its form is PREC=xy. The 'xy' is a two character value composed of two of the following symbols. The type (integer or floating point) of 'opnd1' and 'opnd2' must not be the same. S one word integer (or immediate data) D two word integer F standard precision floating point * use default precision (i.e. PREC = F on left, S on right)

Examples:

```
FPCONV  A,B,COUNT=5,PREC=FD
FPCONV  X,L4,PREC=DF
FPCONV  (6,#1),C
FPCONV  (X,#1),(Y,#2),PREC=DF
```

IBM Internal Use Only

4.3.45 FREEMAIN

Function

The FREEMAIN memory command is used to release a region of memory allocated with the GETMAIN command.

Syntax

label FREEMAIN loadpoint,KEY=,DATA,ERROR=,P1=,P2=,P3=

Required	loadpoint,KEY=,DATA,error
Defaults	None
Indexable	None

Operands

loadpoint	The label of a data area that contains the address of the region to be freed. (P1=)
KEY=	The key value which specifies a partition, is used in conjunction with the loadpoint in locating the DATA region to be released. (P2=)
DATA	The DATA parameter indicates that the region to be freed belongs to non-executable data not a program.
ERROR=	The starting address of the error handling routine to be invoked if the load point given cannot be found. (P3=)

After the instruction is executed, the return code is stored in the first word of the user's TCB. (See also "GETMAIN" on page 88 and "TCBGET" on page 161).

Return Code	Description
0	successful
-1	program load point not found

Example:

```
      FREEMAIN  LOCATION,KEY=2,DATA,ERROR=label
      .
LOCATION DATA F'0'      * address of block to be freed
```

4.3.46 FSUB**Function**

Signed floating point subtraction of opnd2 from opnd1 (opnd1-opnd2).

Syntax

```

label    FSUB    opnd1,opnd2,RESULT=,
              P1=,P2=,P3=

Required  opnd1,opnd2
Defaults  none
Indexable opnd1,opnd2,RESULT

```

Operands

opnd1 The name of the data area from which opnd2 is subtracted. Opnd1 cannot be a self-defining term (constant or equate). The result of the operation is stored in opnd1 unless RESULT is coded. (P1=)

opnd2 The value to be subtracted from opnd1. May be a self-defining term (constant or equate) or the label of a data area. If opnd2 is the address of a data area then the data at that address is assumed to be a signed floating point number of standard precision (double word). If opnd2 is immediate data it may be coded as an integer value between -32768 and 32767 and will be converted to a floating point number prior to performing the operation. (P2=)

RESULT= The name of a data area where the result is to be placed. If not specified, the result is placed in opnd1. (P3=)

See "Floating Point Arithmetic Instructions" on page 23 for a more detailed explanation and examples.

After the operation is performed the return code is stored in the first word of the task control block (see "TCBGET" on page 161). Valid return codes include:

Return Code	Description
-1	successful
1	floating point overflow
5	floating point underflow
7	Hardware check. The APU failed to complete the execution of the command.

Note: The software registers (#1 and #2) may **not** be used as operands in floating point operations since they are only 16 bits in length. The registers may, of course, be used to specify the address operands.

IBM Internal Use Only

4.3.47 GETMAIN

Function

The GET MAIN memory command reserves a block of memory intended for use as storage of non-executable DATA. It is the responsibility of the user to ensure that any region of memory allocated with a GETMAIN command is released using the FREEMAIN instruction.

Syntax

label GETMAIN pagecount,location,error,KEY=,P1=,P2=,P3=,P4=

Required	pagecount,location,error,KEY=
Defaults	None
Indexable	None

Operands

pagecount	The pagecount tells GETMAIN exactly how much room is needed to load the DATA in units of 128 word pages. This value must be a constant and less than 128 pages. (P1=)
location	The label of a 2 word block specified by the user but used by GETMAIN to return the load point and the partition number obtained. (P2=)
KEY=	If this parameter is absent or set to 0 then the search for a region of memory is carried out through all partitions. If only one particular partition will satisfy the requirements then the KEY= operand is used to restrict the search to only that partition. The key value can be an address or a constant between 1 and 3. (P4=)
error	The starting address of the error handling routine to be invoked if the pagecount given is invalid or if the room requested is not available. (P3=)

After the instruction is executed, the return code is stored in the first word of the parameter location.

Return Code	Description
-----	-----
load address	successful
x'00FF'	no space available
0	invalid page count

Example:

```
GETMAIN 2,label,ERROR,KEY=$KEY
      .
      .
ERROR PRINTTEXT 'ERROR DURING GETMAIN@'
```

4.3.48 GETTIME

Function

GETTIME will cause the contents of the system time-of-day clock to be inserted into a three word array in the user program. The three words will contain hours, minutes, and seconds, respectively in binary. It can also obtain the DATE (month, day, and year) provided DATE=YES is present and a six word array has been allocated. This command is can be used to store the time of day and date with data when it is collected. The maximum time is 23.59.59. At midnight, the time-of-day clock is reset to zero and the day is incremented by one.

Note: Day, month and year are incremented and reset as necessary by the supervisor.

Syntax

label GETTIME loc,DATE=,Pl=

Required	loc
Defaults	DATE=NO
Indexable	loc

Operands

loc

The address of

1. a three word table in which the time of day will be stored as hours, minutes, and seconds, (in binary) or
2. a six word table in which the time of day and the date will be stored as hours, minutes, seconds, month, day, and year (in binary). (Pl=)

DATE=

Code DATE=YES to obtain the date as well as the time of day.

Note: Initialization of time and date are NOT the responsibility of the nucleus. time and date can be set by the \$T utility or by a TIME command (See "TIME" on page 163).

IBM Internal Use Only

4.3.49 GETVALUE

Function

GETVALUE is used to read one or more numeric values entered by the terminal operator. The values may be decimal or hexadecimal, of single or double precision, integer or floating point. The printing of an associated prompting message may be unconditional, or it may be conditional upon the absence of advance input. The GETVALUE command will act as a no-op if there is a PROMPT=COND keyword and NO prompt message present and NO advanced input present.

Syntax

```
label    GETVALUE loc,pmsg,count,MODE=,PROMPT=,FORMAT=,
          TYPE=,SKIP=,LINE=,SPACES=,TAB=,P1=,P2=,P3=
```

Required Defaults loc
 MODE=DEC,PROMPT=UNCOND,count=1 (word)
 FORMAT=(6,0,I),TYPE=S
 SKIP=0,LINE=current line,
 SPACES=0,TAB=current column
 If nline is not specified, then it is
 determined by the terminal margin settings.
 Indexable loc,pmsg,SKIP,LINE,SPACES,TAB

Operands

loc Name of the variable to receive the input value. If the number of values requested is greater than one, then multiple values are stored in successive words or doublewords. (P1=)

pmsg Name of a TEXT statement or an explicit text message enclosed in apostrophes. This defines the prompting message which will be issued according to the value of the PROMPT keyword. (P2=)

count Specify the number of values to be entered. To specify double-precision input, code this operand according to the count field convention described under "Data Movement, Logical and Arithmetic Functions" on page 18. With conditional prompting in effect, the absence of advance input causes the prompt message to be issued. Once a prompt message has been issued, zero or more values must be entered. Omitted values leave the corresponding internal variables unchanged. Permitted delimiters between values are the characters slash, comma, period, or blank. (P3=)

MODE= Use MODE=HEX for hexadecimal input. The default (MODE=DEC) is decimal.

SKIP=

LINE=

SPACES=

TAB= See "PRINTTEXT" on page 126.

PROMPT= Code PROMPT=COND or (the default) PROMPT=UNCOND.

FORMAT=(w,d,t)

This parameter is used to specify external format for the input of a single value. The 'count' parameter is ignored. The format is specified as a three element list (w,d,f), defined as follows:

- W A decimal value equal to the maximum field width in bytes expected from the terminal.
- d A decimal value equal to the number of bytes to the right of an assumed decimal point. (An actual decimal point in the input will override this specification.) For integer variables, code this value as zero.

IBM Internal Use Only

t Type of input data
I integer
F floating point F format
E floating point E format

TYPE=

Use this operand only in conjunction with FORMAT=.

S single precision integer (1 word)
D double precision integer (2 words)
F standard precision floating point (2 words)

SKIP=
LINE=
SPACES=
TAB=

These parameters may be used to specify the location within the logical page at which input is to begin, if that location differs from the current line and indent. See "PRINTTEXT" on page 126 for the discussion of these parameters.

Examples:

```
GETVALUE DATA,MESSAGE
GETVALUE DATA1
GETVALUE DATA2,'@ENTER A: ',PROMPT=COND
GETVALUE DATA3,MSG,5,MODE=HEX
GETVALUE DATA10,'@ENTER DATA @',MODE=DEC,FORMAT=(10,5,F),TYPE=F
GETVALUE DATA,'@ENTER B=',MODE=DEC,FORMAT=(9,3,E),TYPE=F
MESSAGE TEXT 'ENTER YOUR AGE'
MSG TEXT 'DATA: '
```

Note:

1. When used with the 3101 display, a GETVALUE command will force the task issuing it to enter the wait queue for the duration of the data transfer. During this time, a task with a lower priority may become active.
2. The unused part of the input field will be unchanged.
3. If a count parameter but not a message is used then two consecutive commas must be used as a delimiter:

```
GETVALUE DATA3,,5,MODE=HEX
```

IBM Internal Use Only

4.3.50 GOTO

Function

GOTO is an unconditional branch to another command or a list of commands from which a selection is made as a function of a specified index value (Computed GOTO).

Syntax

label GOTO loc,P1=

label GOTO (loc0,loc1,loc2,...,locn),index,P1=P2=

Required	loc
Defaults	None
Indexable	index

Operands

loc
The address of the command to be executed after the unconditional branch. If 'loc' is enclosed in parenthesis, the GOTO is 'indirect'. Therefore, the next command is determined by the contents of 'loc'. (P1=)

loc0
The address of the command to be executed if the 'index' value for a Computed GOTO is not in the range 1 to 'n'.

loc1,loc2,...,locn

A list of command addresses. The address selected will be a function of the value of the 'index' field.

index
The address of an 'index' variable whose value is to be used to select the target address for the branch. (P2=)

Examples:

GOTO	LOC0	Branch to LOC0
GOTO	(LOC1)	Branch to location defined by LOC1
GOTO	(ERR,L1,L2),I	Computed GOTO based on value of 'I'. If I is 1, branch to L1. If I is 2, branch to L2. Otherwise, branch to ERR.

4.3.51 IF

Function

IF determines whether a statement is true or false, and then branches to a user specified address or passes control to true code or false code within the IF-ELSE structure.

Syntax

```
label    IF  statement<,THEN>
label    IF  statement,GOTO,loc

Required      One relational statement
Defaults      None
Indexable     data1 and data2 in each relational statement
```

Operands

```
statement    A statement indicating the relationship to be tested. If
              GOTO is coded and the statement is true, the next command
              executed is defined by 'loc', otherwise, execution proceeds
              sequentially. If GOTO is not coded, THEN is assumed and the
              next command is determined by the IF-ELSE-ENDIF structure.

THEN          An optional operand which only serves to comment the command
              for program readability.

GOTO          If the statement is true, control is passed to the command
              at 'loc'. If the statement is false, execution proceeds
              sequentially.

loc           Used with GOTO to specify the address of the command to be
              executed if the statement is true.
```

The description of the syntax and the examples of relational statements can be found in "Program Sequencing" on page 26.

Examples of IF, ELSE, and ENDIF

1. IF with GOTO


```
          IF  (A,EQ,B),GOTO,ANEB
```
2. Single IF


```
          IF  (C,NE,D)
            :
            : (execute if C NE D)
            :
          ENDIF
```
3. IF with ELSE


```
          IF  (#1,EQ,1)
            :
            : (execute if #1 EQ 1)
          ELSE
            :
            : (execute if #1 NE 1)
          ENDIF
```
4. IF with nesting

IBM Internal Use Only

x1	IF (A,EQ,B)	If A equals B and X is
	:	greater than Y, statements
x2	IF (X,GT,Y)	x1, x2, and x3 will be executed.
	:	If A equals B, but X is not
x3	ENDIF	greater than Y, statements x1
	:	and x3 are executed. If A does
x4	ELSE	not equal B, only statement x4
	:	is executed.
	ENDIF	

5. Double IF with ELSE

```
IF (A,EQ,B),AND,(C,EQ,D)
:
: (execute if A EQ B and C EQ D)
ELSE
:
: (execute if either A NE B or C NE D)
ENDIF
```

Note: When a task with a higher priority is waiting in the ready queue while a IF instruction with the count parameter greater than 255 is executing, a task switch will occur before the instruction is completed. The execution of the IF instruction will be resumed after the higher priority task terminates or enters a wait state.

IBM Internal Use Only

4.3.52 INTBIT - DSC

Function

INTBIT is used to define the interrupt bits of the Block Interface Cards which are to be referenced symbolically in the application program. One INTBIT command is required for each bit.

Syntax

INTBIT	INTx,BLOCK=,BIT=,ECB=
Required	INTx,BLOCK=,BIT=
Defaults	None

Operands

INTx=	'x' specifies a symbolic reference number used within the application. This number must be between 1 and 99.
BLOCK=	Specify the single digit hexadecimal address of the DIS block. This address must be between 0 and F.
BIT=	Specify the single digit hexadecimal address of the interrupt bit on the Block Interface Card. This address must be between 8 and F.
ECB=	Specify the label of the event control block to be posted when the interrupt occurs.

Note:

1. The ECB is posted each time that the interrupt line coming into the BIC changes state and the nucleus does not attempt to reset the interrupt status of the device that generated the interrupt. The nucleus merely posts the ECB and masks off the interrupt. Thus the user is responsible for resetting the interrupt status on any devices that require it. If, in the process of resetting the device, the user causes the interrupt line to change state (this is usually the case) a second interrupt will be generated and the ECB will be posted again.
2. It is the user's responsibility to ensure that the proper bit is 'wired-in' on the BIC.
3. Each INTBIT command in the application program causes ENDPROG to generate an IDB. The IDB contains the ECB associated with the interrupting bit on the DIS, and may be referenced using the INTx label as specified in the INTBIT command. For example,

WAIT INT66

will cause a 'wait' for the interrupt line associated with INT66 to change levels at the DIS interface.

IBM Internal Use Only

4.3.53 INTBIT - ARIEL

Function

INTBIT is used to define the interrupt bits of the Block Interface Cards which are to be referenced symbolically in the application program. One INTBIT command is required for each bit.

Syntax

```
INTBIT      INTx,BLOCK=,BIT=,ECB=

Required    INTx,BLOCK=,BIT=
Defaults    None
```

Operands

```
INTx=       Provides a unique symbolic label for the IDB structure this
              statement creates in the application program. The label is
              characterized with a suffix number 1 to 99, for example: INT1
              to INT99.

BLOCK=       The hexadecimal address of the DIS block. The address must
              be between 0 and 1F.

BIT=         Specify the single hexadecimal digit which defines the in-
              terrupt bit to be support by this IDB. There are 8 interrupts
              per block, numbered from 8 through F.

ECB=         Specify the label of an even control block (ECB) to be posted
              when the interrupt occurs. This operand is optional and it
              allows the programmer to define an additional ECB for the
              IDB. When an interrupt occurs for this IDB, both ECBs are
              posted. The ECB is called the general ECB in some documen-
              tation.
```

Note:

1. An interrupt occurs when the interrupt line coming into BIC changes state. When the interrupt occurs the operating system will do the following:
 - reset the interrupt by writing to the Reference Register the contents of the Interrupt Register
 - determine which bits caused the interrupt
 - post all ECBs waiting on the interrupt bits which caused the interrupt
2. The user is responsible for resetting the interrupt status on any devices that require it.
3. If, in the process of resetting the device, the user causes the interrupt line to change state (this is usually the case) a second interrupt will be generated and the ECB will be posted again.
4. The address of the interrupt bit used in the BIT= operand is defined by the slot number of the card which is to generate the interrupt.
5. Each INTBIT command in the application program causes ENDPROG to generate an IDB. The IDB contains the ECB associated with the interrupting bit on the DIS, and may be referenced using the INTx label as specified in the INTBIT command. For example,

WAIT INT66

will cause a 'wait' for the interrupt line associated with INT66 to change levels at the DIS interface.

4.3.54 INTIME**Function**

INTIME provides the user with two forms of interval timing information. The first, 'reltime', is a two word block in the user's program where, the elapsed time since system IPL is stored each time an INTIME is executed. This count is expressed in milliseconds and is in double precision integer format. The maximum value for 'reltime' will be reached in approximately 49 days of continuous operation at which the counter will then roll over to zero. The second, 'loc', is a single precision integer variable where the time in milliseconds since the previous execution of an INTIME instruction is stored. The maximum interval between calls to INTIME (i.e., maximum value that can be stored at 'loc') is 65535 milliseconds or 65.535 seconds.

Syntax

label	INTIME	reltime,loc,INDEX,P2=
Required		reltime,loc
Defaults		No indexing
Indexable		loc

Operands

reltime	The address of a two word table where a 'relative time marker' may be stored. (This field may be defined with a DATA command.) The 'relative time marker' is a double precision count (in milliseconds) which indicates the time relative to the system IPL. It should be initialized to 0.
loc	Proper use of this parameter allows one to measure different intervals from the same origin in time. Buffer address or location where time interval between the last INTIME and the current one is to stored. When 'reltime' = 0, as after initialization, the first interval returned will also be zero. (P2=)
INDEX	Automatic indexing is to be used. ('loc' must be defined by a BUFFER statement when INDEX is used.) If the buffer is full, a return code of X'66' is placed in the first word of the task control block (See "TCBGET" on page 161).

IBM Internal Use Only

4.3.55 IOCB

Function

The IOCB statement defines the console configuration parameters to be used by the ENQT command.

Syntax

label IOCB PAGESIZE=,OVFLINE=,NHIST=

Required	None
Defaults	See discussion below.
Indexable	None

Operands

PAGESIZE=	This operand is as defined for the TERMINAL statement. Its default is the value assigned on that statement during System Generation.
OVFLINE=	As defined for TERMINAL or defined for the System Generation commands.
NHIST=	As defined for TERMINAL or defined for the System Generation commands. (note that NHIST must be less than PAGESIZE.)

Note: IOCB is NOT an executable statement and should therefore be placed in a data area of a program, not in the middle of executable commands.

4.3.56 IOR

Function

Logical 'or' of opnd2 to opnd1.

Syntax

label	IOR	opnd1,opnd2,count,RESULT= P1,P2=,P3=
Required		opnd1,opnd2
Defaults		count=1,RESULT=opnd1
Indexable		opnd1,opnd2,RESULT

Operands

opnd1	The name of the variable to which the operation applies; it cannot be a constant. If the RESULT= operand is not specified, then opnd1 is also the implicit result. (P1=)
opnd2	This operand determines the value by which the first operand is modified. Either the name of a variable or an explicit constant may be specified. (P2=)
count	The number of consecutive variables upon which the operation is to be performed. (P3=)
RESULT=	This operand may optionally be coded with the name of a variable or vector in which the result is to be placed. In this case, the variable specified by the first operand is not modified.

See "Data Movement, Logical and Arithmetic Functions" on page 18 for a more detailed explanation and examples.

IBM Internal Use Only

4.3.57 LEAVE

Function

LEAVE causes a branch to the end of the associated SELECT block. This command may be used more than once in the same SELECT-WHEN block.

Syntax

label	LEAVE
Required	None
Defaults	None
Indexable	None

Operands

None

See "SELECT" on page 145 for examples.

4.3.58 LN

Function

Natural logarithm (base e) of opnd2.

Syntax

label	LN	opnd1,opnd2,P1=,P2=
Required		opnd1,opnd2
Defaults		none
Indexable		opnd1,opnd2

Operands

opnd1	The name of a data area where the result is to be stored.
opnd2	The quantity on which to operation is to be performed. May be an address or an immediate value (constant). If opnd2 is an address then the data at that address is assumed to be a signed floating point number of standard precision (double word). If opnd2 is immediate data it may be coded as an integer value between -32768 and 32767 and will be converted to a floating point number prior to performing the operation.

See "Derived Floating Point Functions" on page 22 for a more detailed explanation and examples.

After the operation is performed the return code is stored in the first word of the task control block (see "TCBGET" on page 161). Valid return codes include:

Return Code	Description
-----	-----
-1	successful
1	Operand too large, or overflow in operation
3	Divide-by-zero
5	Operand too small, or underflow in operation
7	Hardware check. The APU failed to complete the execution of the command.

Note: The software registers (#1 and #2) may not be used as operands in floating point operations since they are only 16 bits in length. The registers may, of course, be used to specify the address of operands.

IBM Internal Use Only

4.3.59 LOAD

Function

The LOAD command is used by a program to load datasets from disk or diskette to Hazel memory. The load support allocates enough Hazel memory to accommodate the data or program to be read in off disk. The load can be made to any one of the partitions of HAZEL4 memory. The user has the option to load either a program, data or a nucleus.

Data parameters may be passed from the LOADING program to the one being LOADED. Also, the LOADING program may synchronize its own execution with that of the LOADED program.

Information being loaded does not have to be a program intended to be executed. It can be a collection of data required by an executing task.

A program may be loaded as an independent program in its own region. The advantages of the 'independent' LOAD operation are:

1. that main storage (a region) is allocated only when required and is otherwise available for use by other programs and
2. that more than one program may be LOADED for simultaneous execution by the LOADING program.

The task code word may be tested to determine the result of the program load operation. The code word is referenced by the taskname. A list of the error codes returned during the load process are listed after the description of the operands.

Syntax

label	LOAD	dscb,MODE=,LOADPT=,ERROR=,PARMNAM=, EVENT=,KEY=,LOGMSG=,P1=,P2=,P3=
Required		dscb
Defaults		MODE=PRG KEY=first partition with sufficient space LOGMSG=YES

Operands

dscb	The label on a DSCB statement which contains the name of the program to be loaded. The DSCB can also contain a optional volume id to specify where in the library the program is to be loaded from. (P1=)								
MODE=	Indicates the type of load. MODE= may be coded as either: <table><tbody><tr><td>PRG</td><td>load program and start</td></tr><tr><td>DATA</td><td>load data</td></tr><tr><td>NOEXEC</td><td>load program and do not start</td></tr><tr><td>NUC</td><td>load nucleus (software reboot)</td></tr></tbody></table>	PRG	load program and start	DATA	load data	NOEXEC	load program and do not start	NUC	load nucleus (software reboot)
PRG	load program and start								
DATA	load data								
NOEXEC	load program and do not start								
NUC	load nucleus (software reboot)								
LOADPT=	The label of the field used by the loader to store the load point (starting location) of the program. This field must be either 2 or 3 words long depending on the application. If a program is to be loaded then the field is 2 words and the partition number is placed in word 2. If a DATA load is to take place then a third word is needed to contain the size in units of 128 word blocks.								
LOGMSG=	Set to YES or NO to indicate whether a 'PROGRAM LOADED' message is to be displayed on the system terminal.								
ERROR=	The label of the first instruction of the routine to be invoked if an error condition occurs during the LOAD process. If not specified, control is returned to the instruction								

IBM Internal Use Only

following the LOAD instruction. Errors can be detected by testing the completion code stored in the first word of the TCB (See "TCBGET" on page 161).

PARMNAME= The label pointing to a list of consecutive parameters to be passed to the LOADED program. The length of the list is specified by the PARM parameter in the PROGRAM statement of the LOADED program. The words in the list may be referenced in the LOADED program by the label \$PARMx where 'x' is the word number in the list. The first word is referred to as \$PARM1. The last word of the Program Header in the LOADED program will be the length of the parameter list. If the length is nonzero, the parameter list immediately follows the Program Header. (P2=)

EVENT= The symbolic name of an event to be posted in the LOADING program when the LOADED program issues a PROGSTOP. The LOADED program knows this name because of the EVENT parameter in its PROGRAM statement. If used, the LOADING program must wait for the LOADED program to terminate. This parameter will generate an ECB if one does not exist.

LOADNUC= Specify YES to request the loading of a nucleus.

KEY= Indicates the memory partition into which the program or data is to be loaded. This operand can be a constant between 0 and 3 or the label of an address containing the key value. (P3=)

After the LOAD instruction is executed, the return code is stored in the first word of the user's TCB.

Return Code	Description
0000	Successful
10xx	Initial read failed
4000	Program not found
5000	Not a tool program
8000	Insufficient space for loading
90xx	Cyclic read failed
C000	Program compatibility load error
D000	IDB conflict

The xx bytes indicate the disk error conditions. See disk error for READ and WRITE.

Note:

1. The LOAD instruction should not be used within an ATTNLIST routine.
2. The LOAD instruction will force the task issuing it to enter the wait queue for the duration of the data transfer. During this time, a task with a lower priority may become active.

IBM Internal Use Only

4.3.60 LOADH

Function

The load from host command is used by programs to load datasets from the host (SERIES/1). The load support allocates enough Hazel memory to accommodate the downloaded data or program. The load can be done into any one of the partitions of HAZEL4 memory. The user has the option to load either a program, data or a nucleus.

Data parameters may be passed from the LOADING program to the one being LOADED. Also, the LOADING program may synchronize its own execution with that of the LOADED program.

Information being loaded does not have to be a program intended to be executed. It can be a collection of data required by an executing task.

A program may be loaded as an independent program in its own region. The advantages of the 'independent' LOAD operation are:

1. that main storage (a region) is allocated only when required and is otherwise available for use by other programs and
2. that more than one program may be LOADED for simultaneous execution by the LOADING program.

The task code word may be tested to determine the result of the program load operation. The code word is referenced by the taskname. A list of the error codes returned during the load process are listed after the description of the operands.

Syntax

label	LOADH	program,PARMNAM=(no)EXEC,DATA,LOADPT=,LOGMSG=, ERROR=,EVENT=,LOADNUC=YES,KEY=,P1=,P2=,P3=
Required		program
Defaults		DATA=no EXEC=yes KEY=first partition with sufficient space LOADNUC=NO LOGMSG=YES

Operands

program	The name of the program (1 to 8 byte EBCDIC name) from the host. (P1=) The program must be named Txxprogram (where xx is the line number of the DSS or 99) on the Series/1 disk. If Txxprogram doesn't exist at the host then T99program will be sent, if available.
EXEC	Indicates whether to execute the program once loaded. The attach code is -2 (X'FFFE').
DATA	Informs the loader that the information to be loaded is not a program but just data.
LOADPT=	The label of the field used by the loader to store the load point (starting location) of the program. This field must be either 2 or 3 words long depending on the application. If a program is to be loaded then the field is 2 words and the partition number is placed in word 2. If a DATA load is to take place then a third word is needed to contain the DATA size in units of 128 word blocks.
LOGMSG=	Set to YES or NO to indicate whether a 'PROGRAM LOADED' message is to be displayed on the system terminal.
ERROR=	

IBM Internal Use Only

The label of the first instruction of the routine to be invoked if an error condition occurs during the LOAD process. If not specified, control is returned to the instruction following the LOAD instruction. Errors can be detected by testing the completion code stored in the first word of the TCB (See "TCBGET" on page 161).

PARMNAME=

The label pointing to a list of consecutive parameters to be passed to the LOADED program. The length of the list is specified by the PARM parameter in the PROGRAM statement of the LOADED program. The words in the list may be referenced in the LOADED program by the label \$PARMx where 'x' is the word number in the list. The first word is referred to as \$PARM1. The last word of the Program Header in the LOADED program will be the length of the parameter list. If the length is nonzero, the parameter list immediately follows the Program Header. (P2=)

EVENT=

The symbolic name of an event to be posted in the LOADING program when the LOADED program issues a PROGSTOP. The LOADED program knows this name because of the EVENT parameter in its PROGRAM statement. If used, the LOADING program must wait for the LOADED program to terminate. This parameter will generate an ECB if one does not exist.

LOADNUC=

Specify YES to request the loading of a nucleus.

KEY=

Indicates the memory partition into which the program or data is to be loaded. This operand can be a constant between 0 and 3 or the label of an address containing the key value. (P3=)

After the LOADH instruction is executed, the return code is stored in the first word of the user's TCB.

Return Code	Description
0000	Successful
20xx	Initial send failed
30xx	Initial receive failed
4000	Invalid program name
5000	Program not found
6000	Not a tool program
7000	Disk error (S/I)
8000	Loader2 not loadable error
9000	Insufficient space for loading
A0xx	Cyclic send failed
B0xx	Cyclic receive failed
C000	Record# mismatch (4 retries)
D000	Program compatibility load error
E000	IDB conflict
E100 - ARIEL	GETMAIN for IDB list failed
E200 - ARIEL	AP/DIS poll list update command error
E300 - ARIEL	AP/DIS not responding

The xx bytes indicate communication error conditions. See communication errors for SEND and RECEIVE.

Note:

1. The LOADH instruction should not be used within an ATTNLIST routine.
2. The LOAD instruction will force the task issuing it to enter the wait queue for the duration of the data transfer. During this time, a task with a lower priority may become active.

IBM Internal Use Only

4.3.61 LOG

Function

Common logarithm (base 10) of opnd2.

Syntax

label	LN	opnd1,opnd2,P1=,P2=
Required		opnd1,opnd2
Defaults		none
Indexable		opnd1,opnd2

Operands

opnd1	The name of a data area where the result is to be stored.
opnd2	The quantity on which to operation is to be performed. May be an address or an immediate value (constant). If opnd2 is an address then the data at that address is assumed to be a signed floating point number of standard precision (double word). If opnd2 is immediate data it may be coded as an integer value between -32768 and 32767 and will be converted to a floating point number prior to performing the operation.

See "Derived Floating Point Functions" on page 22 for a more detailed explanation and examples.

After the operation is performed the return code is stored in the first word of the task control block (see "TCBGET" on page 161). Valid return codes include:

Return Code	Description
-1	successful
1	Operand too large, or overflow in operation
3	Divide-by-zero
5	Operand too small, or underflow in operation
7	Hardware check. The APU failed to complete the execution of the command.

Note: The software registers (#1 and #2) may **not** be used as operands in floating point operations since they are only 16 bits in length. The registers may, of course, be used to specify the address of operands.

IBM Internal Use Only

4.3.62 LSA - DSC

Function

LSA (Logical Space Address) statements are used to define the macrofunctions to be used by the application program.

Syntax

LSA LSAx,BLOCK=,MACRO=,BITS=(u,v),TYPE=
Required LSAx,BLOCK=,MACRO=
Defaults If the (Master) U,V bits are not specified, a full word
 or full byte, depending on the CTC, will be transferred.

Operands

LSAx 'x' specifies a symbolic reference number used within the
 application program. Range = 1-99.
BLOCK= The single digit hexadecimal address of the DIS block. The
 address must be between 0 and F.
MACRO= The single digit hexadecimal address of the macrofunction
 card. The address must be between 1 and F. (location 0 is
 reserved for the BIC).
BITS=(u,v) Indicates a portion of the data bits starting at bit u for
 a length of v. The range for u is 0-15; range for v is 1-16.
TYPE= The TYPE= parameter is only provided for documentation. It
 is not used by the system support and is optional. The fol-
 lowing list of macrofunction card names are available.

AI	Analog Input card
AO	Analog Output card
DI	Digital Input card
DO	Digital Output card
EDS	Electrostatic Diaphragm Switch card
KEYBD	Keyboard card
MD	Magnet Driver
OPMON	Operation Monitor card
PA	Photo Amplifier card
RAM	Random Access Memory card
ROM	Read Only Memory card
RS232C	RS232C card
SD	Solenoid Driver
STEPM	Stepper Motor card
TIMER	Timer card
VIDEO	Video card

Each LSA command in the application program causes ENDPROG to generate an MDCB. The MDCB contains the information necessary to allow the supervisor to control the associated macrofunction according to the DISREAD/DISWRITE commands. The label used on the MDCB is LSAx as specified in the LSA command. The word at this label is used as the data input/output area (if loc is not specified) or as a pointer to the data input/output area (if loc is specified). The application program may reference this word using the LSAx symbol.

Note:

1. It is the user's responsibility to ensure that the proper macrofunction is plugged into the proper location on the DIS.
2. Each BIC can address 16 positions, where 0 is the BIC itself. Up to 16 BICs can be chained.
3. The BITS= Parameter is only valid for use with the DISREAD command. See "DISREAD - DSC" on page 54 and "DISWRITE - DSC" on page 57 for examples.

IBM Internal Use Only

4. LSA is not an executable statement and therefore should be coded in the data area of your program, not in the middle of executable commands.

IBM Internal Use Only

4.3.63 LSA - ARIEL

Function

LSA (Logical Space Address) statements are used to define the macrofunctions to be used by the application program.

Syntax

```
LSA      LSAx,BLOCK=,MACRO=,LSA=(block,macro),TYPE=
Required LSAx
          BLOCK=,MACRO= or LSA=(block,macro)
```

Operands

LSAx
Provides a unique symbolic label for the LSA structure this statement creates in the application program. The label is characterized with a suffix number 'x' = 1 to 99, for example: LSA1 to LSA99.

BLOCK=
The hexadecimal address of the DIS block. The address must be between 0 and 1F.

MACRO=
The single digit hexadecimal address of the macrofunction card. The address must be between 1 and 7. For system level applications, address 0 can be used to communicate with the BIC.

LSA=(block,macro)
LSA address is specified in decimal and in two fields. The block number ranges from 0 to 31 while the macro number ranges from 1 to 7. For system level applications, address 0 can be used to communicate with the BIC.

TYPE=
The TYPE= parameter is only provided for documentation. It is not used by the system support and is optional. The following list of macrofunction card names are available.

AI	Analog Input card
AO	Analog Output card
DI	Digital Input card
DO	Digital Output card
EDS	Electrostatic Diaphragm Switch card
KEYBD	Keyboard card
MD	Magnet Driver
OPMON	Operation Monitor card
PA	Photo Amplifier card
RAM	Random Access Memory card
ROM	Read Only Memory card
RS232C	RS232C card
SD	Solenoid Driver
STEPM	Stepper Motor card
TIMER	Timer card
VIDEO	Video card

Each LSA command in the application program causes ENDPROG to generate an MDCB. The MDCB contains the information necessary to allow the supervisor to control the associated macrofunction according to the DISREAD/DISWRITE commands. The label used on the MDCB is LSAx as specified in the LSA command. The word at this label is used as the data input/output area (if loc is not specified) or as a pointer to the data input/output area (if loc is specified). The application program may reference this word using the LSAx symbol.

Note:

1. The BITS= parameter is no longer supported in the LSA statement.
2. It is the user's responsibility to ensure that the proper macrofunction is plugged into the proper location on the DIS.

IBM Internal Use Only

3. Each BIC can address 8 positions, where 0 is the BIC itself. Up to 32 BICs can be chained.
4. The local block has BLOCK address 0 and the remote blocks have BLOCK addresses 1 to 31.
5. LSA is not an executable statement and therefore should be coded in the data area of your program, not in the middle of executable commands.

IBM Internal Use Only

4.3.64 MAPAD

Function

MAPAD is used to move the storage map address to the assigned location.

Syntax

label MAPAD loc,P1=

Required	loc
Default	None
Indexable	None

Operands

loc
is the address location for storing the storage map address.
(P1=)

Note: Refer to the storage map layout in "Storage Map" on page 250.

IBM Internal Use Only

4.3.65 MDCB - ARIEL

Function

MDCB (Macrofunction Data Control Block) statements are used to define the macrofunctions to be used by the application program. See "DISREAD - DSC" on page 54, "DISWRITE - DSC" on page 57 and "Appendix F. Hazel/4 - Hazel/Ariel DIS Support Compatibility" on page 311.

Syntax

label	MDCB	cmd,buf,LSA=,CTC=,COUNT=,BITS=,TYPE=,RESET=,RETRY, CHAIN,P2=,P3=,P4=,P5=,P6=,P7=,P8=
Required	cmd	
Default	COUNT=1,	
Indexable	buf	

Operands

label When placed on the MDCB with the 'cmd' field: HEADER, it defines the MDCB structure for the DISIO command.

cmd Specify one of the following keyword commands:

HEADER - to define MDCB header

READ - for DIS read operation

WRITE - for DIS write operation

Internal DIS Command - see "DIS Internal Commands" on page 114

A useable MDCB structure must have a header section and one or more command sections. The programmer is responsible for defining the complete structure using two or more MDCB statements. The first MDCB must be defined as a HEADER followed by the required read or write command MDCB(s). The program can chain any number of command MDCBs to a header by using the CHAIN parameter. The DISIO command is used to execute the MDCB structure. All the chained command sections of the MDCB are executed in sequence by a DISIO instruction.

buf A label of an appropriate sized buffer for writing or reading data. The size of the buffer must be compatible with the count specified in the command. For odd CTCs, the low order byte of each word in the buffer is sent or received. The READ command clears the upper byte when it stores a byte in the buffer. Whenever this parameter is omitted, P2= must be defined to show a means for setting the buffer address during execution of the program. If buf parameter is not specified, the system uses an internal byte/word field within the MDCB structure. The application program can access this transfer area in the MDCB by using the MDCB's P2= label. Whenever the 'buf' parameter is omitted, P2= must be specified. (P2=)

LSA=(block,macro)

LSA address is specified in decimal and in two fields. The block number ranges from 0 to 31 while the macro number ranges from 1 to 7. For system level applications, address 0 can be used to communicate with the BIC. Whenever this parameter is omitted, P3= must be defined to show a means for setting the buffer address during the execution of the program. (P3=)

CTC=

Specify the single hexadecimal digit which defines the command tag. Range is 1 to F, however command tag 0 is also available for system level programming. Whenever this parameter is omitted, P4= must be defined to show a means for setting the buffer address during the execution of the program. (P4=)

IBM Internal Use Only

COUNT= Specify the number of times the command tag sequence is to be issued. The data will be fetched from or stored into sequential word locations in the buffer for each execution of the command tag sequence. Whenever this parameter is omitted, P5= must be defined to show a means for setting the buffer address during the execution of the program.

When COUNT=1 and CHAIN is not specified, all other operations (including tasks of higher priority) are locked out for the duration of the transfer. As a result, even if EVENT= is specified, the next sequential EDX instruction will not be executed until the DISIO instruction is completed. (P5=)

BITS=(u,v)
BITS=(u,v,rctc)

For a READ command, BITS= is used to specify the portion of bits read to be passed to the user. 'v' bits are selected starting at bit 'u'. The bits appear right justified with all higher order bits set zero.

For a WRITE command, BITS= is used to specify the group of bits to be written. 'v' bits are written starting at bit 'u'. The 'v' low order bits of the word at 'loc' are used. The other bits outside this group are retrieved from the hardware by using the 'rctc' to read them in. The bits to be written are merged with the bits read and the new pattern is then written using the specified 'ctc' parameter. (P7=)

TYPE=

The TYPE= parameter is only provided for documentation. It is not used by the system support and is optional. The list shows the macrofunction card names defined. (P8=)

AI	Analog Input card	DI	Digital Input card
AO	Analog Output card	DO	Digital Output card

RESET=

Specify YES or NO to clear the DIS error log. This operand must be used with the DISIO internal commands. It will be ignored when specified with a READ or WRITE command.

RETRY

When this operand is specified and an I/O error occurs on a DISIO command, the operating system will retry the failing command 16 times. The RETRY operand is ignored if the COUNT is greater than one.

CHAIN

Specifies that this MDCB will be followed immediately by another MDCB, and that they should be chained together. The programmer must ensure that the next statement is a MDCB; the assembler will not detect this error.

P6=

Specifies the label of the buffer key in the MDCB structure. This parameter allows for cross-partition DIS data transfers. Note, that the operating system clears the buffer key field after the MDCB is executed. Therefore, if a cross-partition transfer is intended, the user must move the key value into the MDCB structure before re-executing the same MDCB structure. If the buffer resides in the program partition, the user does not need to specify the key; in this case, the operating system initializes the buffer key field.

4.3.65.1 DIS Error Log

Each of the 256 LSAs has a one word error log:

IBM Internal Use Only

TX/RX ERRORS	TIME-OUTS
--------------	-----------

TX/RX ERRORS - The following errors are logged in this entry: \$DISDXER, \$DISSLER, \$DISLPAR, \$DISRCRC, \$DISRFLG, \$DISRFC.

TIME-OUTS - The following errors are logged in this entry: \$DISSLTO, \$DISDXTO.

See "DISIO - ARIEL" on page 53 for a full list of DIS Channel Errors.

The maximum error count is 255. When the maximum is reached, the error count is no longer incremented.

4.3.65.2 DIS Internal Commands

The DIS Internal Commands are used by the operating system. The following is a list of the DIS Internal Commands:

CARDL (Card List)

This command requires a buffer (see "CARDL Buffer Format" on page 117 for detailed description of the buffer format). Upon completion of this command, the buffer will contain the card ids and LSAs of every macrofunction card in the system (including AP/DIS, Ariel and AP/CSS cards).

EXECRAM This command causes AP/DIS to branch to memory location X'2000'. The EXECRAM command should only be used for hardware diagnostics. The user should patch AP/DIS RAM memory starting at location X'2000' before executing this command. If no code is entered at this location, this command becomes a NOOP.

FCARD (Find Card)

This command requires a buffer (see "FCARD Buffer Format" on page 117 for detailed description of the buffer format). Before this command is issued, the buffer must be initialized with the card id of the macrofunction card to be located. Upon completion of this command, the buffer will contain LSAs of the macrofunction card to be located.

Note that, the FCARD command cannot be used to locate the processor cards (Ariel, AP/DIS or AP/CSS). An alternative method to locate the slot numbers and/or trigger word addresses is shown below:

```

MOVE    #1,0
MOVE    #1,(+NUCDIR,,*)
MOVE    #1,(+NCAPDIR,#1)
MOVE    TRIGADDR,(+equate,#1)
AND     (+equate,#1),X'0007',RESULT=SLOTNUM

```

TRIGADDR - will contain the trigger address of AP/CSS or AP/DIS.
 SLOTNUM - will contain the slot number of AP/CSS, AP/DIS or Ariel card (0-7).
 #equate - #ARIEL, #APDIS or #APCSS.

NOOP This command on its own does not perform any operations. However, when RESET=YES is specified with this command, all error counts in the DIS Error Log are set to zero.

POLLON This command activates DIS interrupt polling.

POLLOFF This command suspends DIS interrupt polling.

READE (Read the DIS Error Log)

This command requires a buffer (see "READE Buffer Format" on page 115 for detailed description of the buffer format). Upon suc-

IBM Internal Use Only

Successful completion of this command, the buffer will contain only the LSAs which have non-zero entries in the DIS Error Log ("DIS Error Log" on page 113).

READP (Read the Interrupt Poll List)

This command requires a buffer (see "READP Buffer Format" for detailed description of the buffer format). Upon successful completion of this command, the buffer will contain the poll list.

UPDATEP (Update the Interrupt Poll List)

This command allows interrupt bits to be added or deleted from the poll list. When all interrupt bits have been deleted from a given poll list entry (a block), that entry is deleted from the poll list. If an interrupt bit is set, and its poll list entry does not exist, the needed entry is created. This command requires a buffer (see "UPDATEP Buffer Format" on page 116 for detailed description of the buffer format). Before this command is issued the buffer must be initialized; the block number(s) and associated address to be added or deleted must be specified. After the completion of the command the return code is #CNDERR at least one of the updates specified in the buffer has failed. In this case, the buffer will contain the error codes associated with failing update requests.

4.3.65.3 DIS Internal Command Buffers

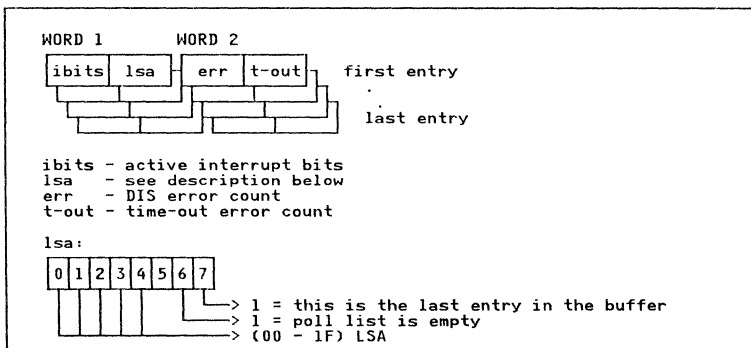
The READP, UPDATEP, READE, CARDL and FCARD commands require buffers for data transfer. The UPDATEP and FCARD commands require that the buffers be initialized before the commands are issued.

READP Buffer Format:

INPUT: none.

OUTPUT: ibits, lsa, err, t-out.

There buffer size must be 64 words (32 blocks times 2 words per block).

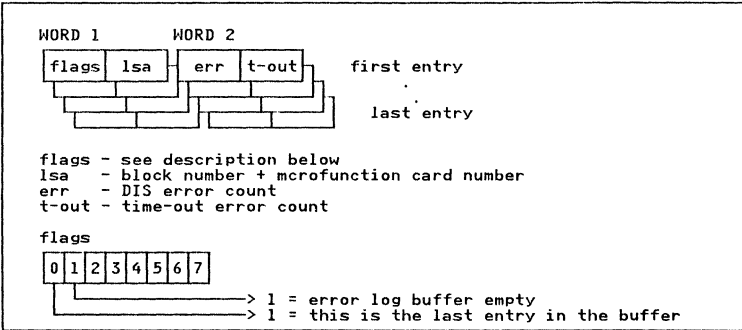


READE Buffer Format:

INPUT: none.

IBM Internal Use Only

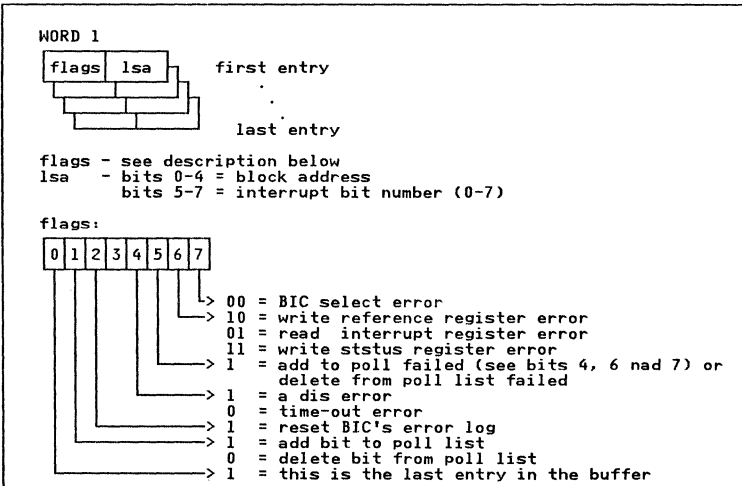
OUTPUT: flags, lsa, err, and tout.



UPDATEP Buffer Format:

INPUT: flags (bits 0, 1 and 2), lsa.

OUTPUT: flags (bits 4, 5, 6, and 7).



Bits 0-2 in the flags field are input bits and bits 3-7 are output bits. The output bits set by the operating system inform the user about the status of the update operations.

When a new block (not just a new interrupt bit) is added to the poll list, the following operations are performed:

1. BIC is selected.

IBM Internal Use Only

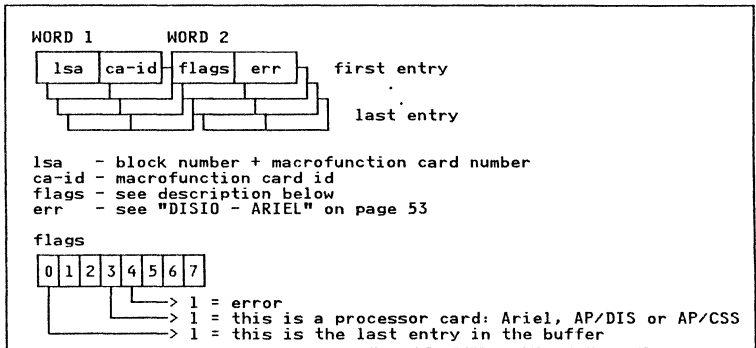
2. Interrupt register is read.
3. Interrupt register is written to the reference register (to reset pending interrupts).
4. Interrupts are enabled.

CARDL Buffer Format:

INPUT: none.

OUTPUT: lsa, ca-id, flags and err.

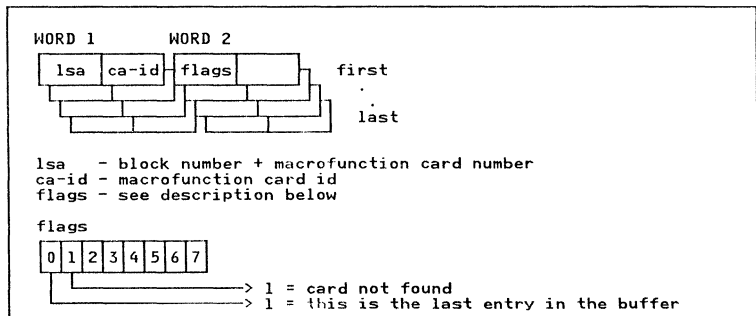
There buffer size must be 512 words (256 LSAs times 2 words per LSA).



FCARD Buffer Format:

INPUT: ca-id (in the first entry only)

OUTPUT: lsa, ca-id and flags.



IBM Internal Use Only

4.3.66 MOVE

Function

Opnd2 is moved to opnd1.

Syntax

label	MOVE	opnd1,opnd2,count,FKEY=,TKEY=
Required		opnd1,opnd2
Defaults		count=1
Indexable		opnd1,opnd2

Operands

opnd1	The name of the variable to which the operation applies; it cannot be a constant. (P1=)
opnd2	This operand determines the value by which the first operand is modified. Either the name of a variable or an explicit constant may be specified. (P2=)
count	The number of consecutive variables upon which the operation is to be performed. (P3=)
FKEY=	The FKEY parameter permits the source operand (opnd2) to be in any one of the memory partitions not only the currently in use. Along with operand 2, FKEY forms a complete address by specifying not only the address of the source operand but also the partition of memory in which it resides. This parameter can be either an constant or an address containing a key value. It cannot however, be indexed in any way. The key values must be between the values of 0 and 3 for Hazel/4 and between the values of 0 and 7 for Hazel/Ariel A value of 0 will not alter the presently active partition. A key validation is not done at the assembly stage but rather during execution of the instruction at which an invalid key will cause the machine to process check. Operand 2 cannot be an constant or an index register if FKEY is used. It may be an address or an indexed value: (0,#1).
TKEY=	The TKEY parameter has the same characteristics as FKEY. It however defines the partition that contains the destination operand (opnd1). If TKEY is used operand 1 cannot be a constant or an index register. Using both FKEY and TKEY allows the transfer of data from any valid partition to another. Any operand with no key associated with it is assumed to reside in the partition that contains the MOVE instruction.

See "Data Movement, Logical and Arithmetic Functions" on page 18 for a more detailed explanation and examples.

Note: When a task with a higher priority is waiting in the ready queue while a MOVE instruction with the count parameter greater than 255 is executing, a task switch will occur before the instruction is completed. The execution of the MOVE instruction will be resumed after the higher priority task terminates or enters a wait state.

4.3.67 MOVEA**Function**

The address of opnd2 is moved to opnd1. The address of opnd2 includes the effects of indexing.

Syntax

```
label    MOVEA    opnd1,opnd2
                    P1=,P2=
```

Required	opnd1,opnd2
Defaults	none
Indexable	opnd1

Operands

opnd1	The name of the variable to which the operation applies; it cannot be a constant. (P1=)
opnd2	This operand determines the address value that is placed in opnd1. (P2=)

See "Data Movement, Logical and Arithmetic Functions" on page 18 for a more detailed explanation and examples.

IBM Internal Use Only

4.3.68 MULTIPLY

Function

Signed multiplication of opnd1 by opnd2. May be abbreviated MULT.

Syntax

label	MULTIPLY	opnd1,opnd2,count,RESULT=,PREC= P1=,P2=,P3=
Required		opnd1,opnd2
Defaults		count=1,RESULT=opnd1,PREC=S
Indexable		opnd1,opnd2,RESULT

Operands

opnd1	The name of the variable to which the operation applies; it cannot be a constant. If the RESULT= operand is not specified, then opnd1 is also the implicit result. (P1=)
opnd2	This operand determines the value by which the first operand is modified. Either the name of a variable or an explicit constant may be specified. (P2=)
count	The number of consecutive variables upon which the operation is to be performed. (P3=)
RESULT=	This operand may optionally be coded with the name of a variable or vector in which the result is to be placed. In this case, the variable specified by the first operand is not modified.
PREC=	This operand takes the form PREC=XYZ, where X applies to opnd1, Y to opnd2, and Z to the result. The value for each specification may be either S (single-precision) or D (double-precision). See "Mixed-precision Operations:" on page 19 for restrictions and abbreviations of the PREC= operand.

See "Data Movement, Logical and Arithmetic Functions" on page 18 for a more detailed explanation and examples.

Note: When a task with a higher priority is waiting in the ready queue while a MULTIPLY instruction with the count parameter greater than 255 is executing, a task switch will occur before the instruction is completed. The execution of the MULTIPLY instruction will be resumed after the higher priority task terminates or enters a wait state.

IBM Internal Use Only

4.3.69 OPEN

Function

The OPEN command identifies the program to the Communications System, and must be the first communication command executed in the program.

If the user wishes to receive data, a receive port must be defined. If the program appears in the communications system table, then receive port 0 is always defined. On the SERIES/1, a program in the system table will be automatically loaded if a message arrives on port 0 when the program is not active. The OPEN command may also specify up to 127 additional numbered ports, which remain defined until a CLOSE command is executed.

Syntax

label OPEN name,PORTS=

Required None

Defaults name= label on PROGRAM statement,PORTS=0

Indexable None

Operands

name

The 6 character name used to identify the program to the Communication System. This is the name to be used as a destination by sending programs.

PORTS=

The number of receive ports required for data sent from other programs (in addition to port 0, if any). This must be a constant between 0 and 127, or an assembler expression.

4.3.70 OTHER

Function

The OTHER statement defines the start of the "false" code associated with the preceding WHEN statements. The "false" code following the OTHER statement is optional. The end of the "false" code is delimited by the ENDSELECT statement. Note, that the ENDSELECT is required even if there is no "false" code.

Syntax

label	OTHER
Required	None
Defaults	None
Indexable	None

Operands

None

See "SELECT" on page 145 for examples.

IBM Internal Use Only

4.3.71 POST

Function

The POST command is used to signal the occurrence of an event.

Syntax

label	POST	event,code,KEY=,P1=,P2=,P3=
Required		event
Defaults		code=-1,KEY=current partition
Indexable		event

Operands

event	The symbolic name of the event. The name may be uniquely defined by the POST or may be defined (1) in an EVENT= operand of another command, (2) with an ECB statement, or (3) in another POST command. (P1=)
code	DIS interrupts are special events which may be simulated with a POST. This is useful when one task is waiting for a macrofunction to interrupt and a second task wishes to activate the first, as in a program termination sequence. In this case, issue a POST INTx where 'x' is the interrupt number as assigned in the INTBIT command. A word to be inserted into a control block for the event. The code word is referred to by the event name and may be used as a flag to indicate a condition or a status. The code cannot be 0 as 0 indicates the event has not yet occurred. If you attempt to post an ECB with a 0 code the 0 will be changed to -1. (P2=)
KEY=	The key value indicates which partition of memory contains the ECB to be posted. This operand is described in the AT-TACH instruction. (P3=)

Note: If the event is not defined elsewhere, this command will generate a new ECB by the name specified in 'event'.

IBM Internal Use Only

4.3.72 PRINDATE

Function

PRINDATE prints the date on the console. The value is printed in the form MM/DD/YY.

Syntax

label	PRINDATE
Required	None
Defaults	None
Indexable	None

Operands

None

Example:

```
PRINTTEXT 'TODAYS DATE IS '  
PRINDATE
```

The date may be set using the TIME command (See "TIME" on page 163) or the \$I supervisor utility (See "Supervisor Utility Functions" on page 173).

Note: When used with the 3101 display, a PRINDATE command will force the task issuing it to enter the wait queue for the duration of the data transfer. During this time, a task with a lower priority may become active.

4.3.73 PRINT

Function

The PRINT statement is used to control printing of the assembly listing.

A program may contain any number of PRINT statements. PRINT statement options remain in effect until the corresponding opposite option is specified.

Syntax

PRINT ON/OFF,GEN/NOGEN,DATA/NODATA

Required	None
Defaults	(Initially) ON,GEN,NODATA

Operands

ON	A listing is printed.
OFF	No listing is printed.
GEN	All statements generated by commands are printed. Note, however, that the post-processor will remove the generated statements and leave only the generated constants in its output listing.
NOGEN	Statements generated by commands are not printed with the exception of MNOTE (error messages) which will print regardless of NOGEN. The command itself will still appear in the listing.
DATA	Generated constants are printed out in full in the listing.
NODATA	Only the leftmost four words are printed on the listing.

Note, that the operands may be specified in any sequence.

IBM Internal Use Only

4.3.74 PRINTTEXT

Function

The PRINTTEXT command is used to write a message to the console. CRT Control is executed before the message is written.

Syntax

label	PRINTTEXT	msg,LINE=,SKIP=,SPACES=,TAB=,MODE=,P1=
Required Defaults		At least one operand LINE=(current line),SKIP=0, SPACES=0,TAB=current column If nline is not specified, then it is determined by the terminal margin settings.
Indexable		msg,LINE,SKIP,SPACES,TAB

Operands

msg	The name of a TEXT statement which defines the message to be printed or an explicit text message enclosed in apostrophes. A 'a' character is a new line character unless MODE=LINE. (P1=)
LINE=	This operand specifies the line number on the CRT at which the skip or write operation will begin. If the value is greater than PAGESIZE then the LINE= is ignored. If the value is less than the current line number, then the screen is blanked.
SKIP=	The number of lines to be skipped before writing the message. If the value specified exceeds PAGESIZE, then the result is a blank screen and the write operation beginning on line NHIST+1.
SPACES=	Code the number of spaces which are to precede the message being written. This operand may be used without an associated message in order to introduce spaces between successive PRINTTEXT or PRINTNUM commands.
TAB=	This operand specifies the column number with respect to the left margin on the terminal screen where the write operation will begin. If the value is greater than the logical line length or less than the current column number, it is ignored. TAB= operand is implemented only on systems with 3101 terminals.
MODE=	Code MODE=LINE if the text includes imbedded 'a' characters which are <u>not</u> to be interpreted as 'newline'.

Examples:

```
PRINTTEXT TEXT1
PRINTTEXT 'aSTART OF PROGRAM'
PRINTTEXT TEXT2,SPACES=4
PRINTTEXT TEXT3,LINE=1,SPACES=2
PRINTTEXT SKIP=1,TAB=3
PRINTTEXT TEXT1,LINE=(+EUQATE,#1),TAB=A
PRINTTEXT ' MESSAGE ',SKIP=(A,#1),SPACES=(8,#2)
```

Note:

1. TEXT is only written to the terminal when the screen buffer (TBUF) is full. Hence a partial line may not appear on the screen due to a

IBM Internal Use Only

PRINTTEXT immediately. To force the line onto the screen, use a '@' as the last character of the message or use MODE=LINE.

2. When used with the 3101 display, a PRINTTEXT command will force the task issuing it to enter the wait queue for the duration of the data transfer. During this time, a task with a lower priority may become active.

IBM Internal Use Only

4.3.75 PRINTIME

Function

PRINTIME prints the time of day on the console. The value printed is in the form HH:MM:SS, according to a 24-hour clock.

Syntax

label	PRINTIME
Required	None
Defaults	None
Indexable	None

Operands

None

Note: The time of day may be set using the TIME command (See "TIME" on page 163) or the \$T supervisor utility (See "Supervisor Utility Functions" on page 173).

Note: When used with the 3101 display, a PRINTIME command will force the task issuing it to enter the wait queue for the duration of the data transfer. During this time, a task with a lower priority may become active.

4.3.76 PRINTNUM

Function

PRINTNUM is used to convert an integer or a floating point variable to a printable decimal or hexadecimal format and write it to the terminal with optional format control. Format specification can include the number of elements per line and the spacing between elements.

Syntax

```
label    PRINTNUM  loc,count,nline,nspace,MODE=,FORMAT=
                        TYPE=,SKIP=,LINE=,SPACES=,TAB=
                        P1=,P2=,P3=,P4=

Required Defaults      loc
                        count=1,nspace=1,MODE=DEC
                        FORMAT=(6,0,I),TYPE=S,
                        SKIP=0,LINE=current line,
                        SPACES=0,TAB=current column
                        If nline is not specified, then it is
                        determined by the terminal margin settings.
Indexable              loc,SKIP,LINE,SPACES,TAB
```

Operands

loc Address of the first value to be printed. Successive values are taken from successive words or doublewords. (P1=)

count The number of values to be printed. To specify double-precision output, use the count field convention described under "Data Movement, Logical and Arithmetic Functions" on page 18. (P2=)

nline The number of values to be printed per line. (P3=)

nspace The number of spaces by which printed values will be separated. (P4=)

MODE= Code MODE=HEX for hexadecimal output. The default is decimal (MODE=DEC).

SKIP=

LINE=

SPACES=

TAB= See the PRINTTEXT operands "PRINTTEXT" on page 126 for the discussion of these parameters.

FORMAT=(w,d,t)

This operand is used to specify the external format of a single variable to be printed. It appears in the form of a three element list (w,d,f), and when specified, count, nline, nspace and MODE are ignored. The format is defined as follows:

- w** A decimal value equal to the field width in bytes of the data to be printed.
- d** A decimal value equal to the number of significant digits to the right of the decimal point. For the integer format, this value must be zero; for the floating point F format, it must be less than or equal to w-2, and for the floating point E format less than or equal to w-6.
- t** Format of the output data
 - I** integer
 - F** floating point F format
 - E** floating point E format

TYPE=

This operand is used to specify the type of the internal variable to be printed. Used only in conjunction with the FORMAT operand.

IBM Internal Use Only

S single precision integer (1 word)
D double precision integer (2 words)
F standard precision floating point (2 words)

SKIP=
LINE=
SPACES=

See "PRINTTEXT" on page 126.

Examples:

```
PRINTNUM A
PRINTNUM BUF1,10
PRINTNUM AX,MODE=HEX
PRINTNUM BUF2,10,5,3
PRINTNUM BZ,(10,DWORD),MODE=HEX
```

Note: When used with the 3101 display, a PRINTNUM command will force the task issuing it to enter the wait queue for the duration of the data transfer. During this time, a task with a lower priority may become active.

IBM Internal Use Only

4.3.77 PROGRAM

Function

PROGRAM is used to define certain basic parameters of a user program to the system. The required information is the name of the main task and the label used for the first command in the program. In addition, the user may define the priority of the main task. Task priorities range from 0 to 255, the highest priority being 0. PROGRAM must be the first statement of every user program.

Priorities separate tasks according to the relative criticality of their real time needs for CPU time. Priority assignments must therefore account for other programs expected to be executing simultaneously.

Syntax

```
taskname PROGRAM start,priority,EVENT=,PARAM=,TERMERR=
```

Required	taskname,start
Defaults	priority=150
Indexable	None

Operands

taskname	The name of the user's main task. A system control block known as the Task Control Block or 'TCB' is generated for each task in an application program. The first word of the TCB is assigned the name specified in the 'taskname' operand. This word is known as the 'task code word' and has a special significance in program operation. For I/O operations it is used to return a completion code to the user. Thus, the taskname may be used in an IF command to test for a successful completion of an I/O operation.
start	The label or address of the first command to be executed in the user's program.
priority	The priority of the program's main task. The range is from 0 (highest priority) to 255 (lowest priority).
EVENT=	EVENT=name is used to name the event which will be posted when the task is detached. It must be defined only if another task will issue a WAIT for this event.
PARAM=	'n' is a word count specifying the length of a parameter list to be passed to this program at load time. Each word in the list may be referenced by the symbolic name \$PARAMx where 'x' is the word position number in the list beginning with 1. The maximum length of this list is 368 words. This parameter is valid for programs to be loaded by a LOADM command. The list address is specified as an operand of that command. The list would be filled in by the LOADING program and there are no restrictions as to its contents.
TERMERR=	Label of the user routine that is to be given control if an unrecoverable I/O error occurs.

Examples:

```
TASK1    PROGRAM  START1
```

The main task is named TASK1 and the first executable command has the label START1. The priority of TASK1 is the default priority, 150.

```
TSAMP2   PROGRAM  BEGIN,200
```

IBM Internal Use Only

The main task, which is named TSAMP2, has a priority 200 and starts at the label BEGIN.

ATASK PROGRAM GOPROG

The main task, ATASK, starts at GOPROG.

4.3.78 PROGSTOP

Function

PROGSTOP is used to terminate execution of a program and release the storage allocated to it. There can be more than one PROGSTOP statement in a program.

Syntax

```
label    PROGSTOP code,LOGMSG=,PI=

Required      None
Defaults     code=-1,LOGMSG=YES
Indexable     None
```

Operands

```
code      The post code to be inserted in the 'event' named in the
          associated TASK or PROGRAM statements. (PI=)

LOGMSG=    Set to YES or NO to indicate whether or not a 'PROGRAM ENDED'
          message is to be displayed on the system terminal.
```

Note: The PROGSTOP instruction is NOT to be used within an ATTNLIST routine.

IBM Internal Use Only

4.3.79 PWR

Function

Opnd1 is raised to the power in opnd2 (opnd1 ** opnd2).

Syntax

label	PWR	opnd1,opnd2,P1=,P2=
Required		opnd1,opnd2
Defaults		none
Indexable		opnd1,opnd2

Operands

opnd1	The name of the data area to be raised to the power in opnd2.
opnd2	The power to which opnd1 is raised. May be an address or an immediate value (constant). If opnd2 is an address then the data at that address is assumed to be a signed floating point number of standard precision (double word). If opnd2 is immediate data it may be coded as an integer value between -32768 and 32767 and will be converted to a floating point number prior to performing the operation.

See "Derived Floating Point Functions" on page 22 for a more detailed explanation and examples.

After the operation is performed the return code is stored in the first word of the task control block (see "TCBGET" on page 161). Valid return codes include:

Return Code	Description
-----	-----
-1	successful
1	Operand too large, or overflow in operation
3	Divide-by-zero
5	Operand too small, or underflow in operation
7	Hardware check. The APU failed to complete the execution of the command.

4.3.80 QCB

Function

QCB generates a Queue Control Block (QCB) which is the symbolic representation of either a system or user resource.

Normally this statement will not be needed for writing application programs. However, it may be used for special purposes such as controlling their location within a program.

Syntax

label	QCB	code
Required		label
Defaults		code = -1
Indexable		None

Operands

code

Code field is the first word in the QCB. The code field may be initialized by the user. A zero in the code field implies that the resource is busy. If the code field is initialized to zero a WAIT will be issued.

Note:

1. QCB is not an executable statement and should therefore be placed in a data area of a program, not in the middle of executable commands. Like ECBs,
2. QCBs should not be coded as consecutive DATA statements (i.e. DATA 5F'0'), and should not be moved with the MOVE command. See "SYSCOPEN" on page 159 and "SYSCDEF" on page 157 for information on how to move QCB's into SYSCOM.

IBM Internal Use Only

4.3.81 QUESTION

Function

QUESTION allows the operator to determine the direction of a conditional branch in the program. The prompt message is printed unconditionally after which the operator may enter Y (Yes) or N (No). Note that advance input for subsequent GEIVALUE and READTEXT commands may follow the response.

Syntax

Label QUESTION pmsg,YES=,NO=,P1=

Required pmsg and either YES= or NO=, or BOTH YES= and NO=
Defaults If one of the operands YES or NO is not
 specified the corresponding response will
 cause the next instruction to be executed.

Indexable pmsg

Operands

pmsg The prompt message, specified either as the name of a TEXT
 statement or as an explicit message enclosed in quotes.
 (P1=)

YES= Label of the command at which execution will continue if the
 answer is 'Yes'.

NO= The label at which execution will continue if the answer is
 'No'.

Examples:

```
QUESTION TEXT3,YES=POINT11
QUESTION 'DO IT AGAIN?',NO=EXIT
QUESTION 'RESTART?',YES=INITIAL,NO=ENDP
```

```
TEXT3      TEXT      'GO TO POINT1? '
```

When used with the 3101 display, a QUESTION command will force the task issuing it to enter the wait queue for the duration of the data transfer. During this time, a task with a lower priority may become active.

IBM Internal Use Only

4.3.82 READ

Function

The READ instruction retrieves one or more records from a disk or diskette and places them in a buffer area you define. You must allocate sufficient buffer space for the operation. Successive sequential access to a dataset is not supported internally and must be supported by the application program by modification of the 'relrecno' parameter. Cross-partition transfers are supported by the READ command through the KEY= parameter.

Syntax

label READ dscb,loc,recnt,relrec,KEY=,WAIT=,END=,ERROR=,PREC=,
P1=,P2=,P3=,P4=,P5=

Required	dscb,loc
Defaults	recnt=1,relrec=1
Indexable	loc,count,relrec

Operands

dscb	The label of a DSCB which contains the dataset's 'name' and possibly the volume id for that dataset. See "DSCB" on page 65.
loc	The label of the buffer area where the data is to be placed. When reading disk or diskette datasets you must make sure that the area is a multiple of 256 bytes.
recnt	The number of contiguous records to be read. If the program sets the field to 0, no I/O operation is performed. A count of the actual number of records read is returned in the second word of the task control block if WAIT=YES is coded. Note, however, if the incorrect block size is specified, the correct block size is stored in the second word of the TCB, not the number of records transferred. If an end_of_data condition occurs (reading beyond the end of the dataset) the transfer ends at the end of the dataset and a end_of_data return code is returned.
relrec	The relative record number within a dataset where a read operation will start. This parameter can be formatted as a self-defining positive decimal value (or equate) or as a label of a variable containing the self-defining value. Note, relative record numbers must start from one (not zero). The S/I uses the relrec=0 to indicate a sequential disk operation; this is currently not supported in the DSC. The POINT command which controls sequential disk operations is currently not supported in the DSC. An EOD indication is returned if an attempt is made to access a record outside of the physical dataset. If you code a self-defining term, or an equated value indicated by a plus sign (+), then it is assumed to be a single-word value and is, therefore, generated as an inline operand. Because this is a one word value, it is limited to a range of 1 to 32767 (x'7FFF'). If you code an indexable value or an address for this operand, the PREC operand can be used to further define whether relrecno is to be a single or double word value. See PREC=.
PREC=	This operand further defines the relrec operand when you code an address or an indexable value for that operand. PREC=S (the default) limits the value of relrec to a single precision value (maximum 32767).
END=	The label of the first instruction of the routine to be invoked if an end of dataset condition is detected during the

IBM Internal Use Only

READ operation (return code=10). If you do not code this operand, the system treats an end of dataset condition as an error. Do not code this operand if you code WAIT=NO.

ERROR=

The label of the first instruction of the routine to be invoked if an error condition occurs during the execution of this operation. If you do not specify this operand, control passes to the instruction following the READ and you must test the return code in the first word of the task control block for errors. Do not code this operand if you code WAIT=NO.

WAIT=

YES (default), to suspend the current task until the operation is complete. NO, to return control to the current task after the operation is initiated. Your program must issue a subsequent WAIT on the label of the DSCB used in the READ command to determine when the operation is complete. You can not code the END and ERROR operands if you code WAIT=NO. You must subsequently test the return code in the Event Control Block (ECB) named in the DSCB or in the first word of the task control block (TCB). The label of the TCB is the label of your program or task. Two return code are significant; -1 indicates a successful end of operation and +10 indicates 'end of dataset' which may be of logical significance and not just an error. For programming purposes, any other return codes should be treated as errors.

Px=

The Px parameters are used to append labels to other parameters in the READ command. The user can then modify these variables using those labels to control the READ operation. Note, that the parameters must be modified in the same sense (ie. value or address) consistent with their original parameter definition. Px ASSIGNMENTS:

P1 -	dscb	address
P2 -	loc	address
P3 -	recnt	value / address
P4 -	relrec	value / address
P5 -	KEY=	value / address

Note: The READ instruction without WAIT=NO parameter specified, will force the task issuing it to enter the wait queue for the duration of the data transfer. During this time, a task with a lower priority may become active.

4.3.83 READTEXT**Function**

This command is used to read an alphanumeric text string entered at the console. The issuance of an associated prompting message may be either unconditional, or conditional upon the presence of advance input. The READTEXT command will act as a no-op if there is a PROMPT=COND keyword and NO prompt message present and NO advanced input present.

Syntax

label	READTEXT	loc,pmsg,PROMPT=,PROTECT=,MODE= SKIP=,LINE=,SPACES=,TAB=,P1=,P2=
Required		loc
Defaults		PROMPT=UNCOND,PROTECT=NO,MODE=WORD SKIP=0,LINE=current line, SPACES=0,TAB=current column If nline is not specified, then it is determined by the terminal margin settings.
Indexable		loc,pmsg,SKIP,LINE,SPACES,TAB

Operands

loc	The name of a TEXT statement defining the storage area which is to receive the data. The storage area may be defined by DATA as well, but the format produced by the TEXT statement must be adhered to. In order to satisfy the receiving count, the input will be either truncated or padded to the right with blanks as necessary. (P1=)
pmsg	The name of a TEXT statement or an explicit text message enclosed in apostrophes. This defines the prompting message which will be issued according to the value of the PROMPT= keyword. (P2=)
PROMPT=	Code PROMPT=COND or PROMPT=UNCOND (the default).
PROTECT=	Code PROTECT=YES if the input text is not to be printed on the terminal. (This operand is effective only for devices which require the processor to 'echo' input data to the output device. It is effective for DEVICE=CONSOLE.)
MODE=	Code MODE=LINE if the string to be read can include blanks or commas. Any portion of the input line which extends beyond the count indicated in the receiving TEXT statement will be ignored and will not be retained as advance input.
SKIP= LINE= SPACES= TAB=	See "PRINTTEXT" on page 126.

Examples:

```

READTEXT  OPTION,'ENTER OPTION: ', PROMPT=COND
READTEXT  NAME,'ENTER YOUR NAME: '
READTEXT  PASSWORD,'ENTER PASSWORD: ',PROTECT=YES
READTEXT  NEXTLINE,,MODE=LINE

```

```

OPTION  TEXT      LENGTH=2
NAME    TEXT      LENGTH=44
PASSWORD TEXT     LENGTH=8
NEXTLINE TEXT     LENGTH=80

```

Note:

IBM Internal Use Only

1. The input field will be blanked before the read.
2. When a READTEXT command executes, it automatically enqueues the terminal. Other tasks are inhibited from immediate use until the terminal is dequeued by pressing the enter key.
3. When used with the 3101 display, a READTEXT command will force the task issuing it to enter the wait queue for the duration of the data transfer. During this time, a task with a lower priority may become active.

IBM Internal Use Only

4.3.84 RECEIVE

Function

The RECEIVE command transfers data sent from another machine into a user specified buffer. Origin information or request data sent with the message and the actual number of words are saved in the RECEIVE command. The return code is also saved in the command.

Messages for port 0 or for all ports if the program is OPEN, will be queued up until a RECEIVE command is executed. Should there be no message already in the queue, the RECEIVE command provides three options:

- wait indefinitely for data to be received (default)
- wait for a specified amount of time (TIME=), or
- exit the command immediately (NOWORK=).

Syntax

label	RECEIVE	buffer,count,PORT=,TIME=,NOWORK=,RCNT=, ORG=,KEY=,RC=,P1=,P2=,P3=,P4=
Required		buffer,count
Defaults		PORT=0,TIME=,NOWORK= (wait indefinitely) KEY=current partition
Indexable		buffer

Operands

buffer	The location of a buffer to hold the received data. (P1=)
count	The maximum number of words the buffer will hold. This may be a number between 0 and 500, or an assembler expression preceded by an '+'. (P2=)
PORT=	The number of a receive port in the program from which data is to be obtained.
TIME=	If no messages are waiting at the receive port, this specifies the maximum amount of time to wait. This is a number between 1 and 65535, or an assembler expression preceded by an '+', giving the timeout in milliseconds. (P3=)
NOWORK=	If no messages are waiting at the receive port, this is the label of the EDX/J instruction at which to resume execution.
RCNT=	A unique label to be attached to the receive count field in this RECEIVE command. After a successful RECEIVE, this field contains the actual number of words transferred.
ORG=	A unique label to be attached to the ORIGIN field in the RECEIVE. After a successful RECEIVE, this field contains the destination block for the program sending the message, or the request data sent with the message. If request was sent, the port number is given as X'FF'
RC=	A unique label to be attached to the return code field in the RECEIVE.
KEY=	This parameter indicates which partition of DSC memory the buffer is located. (P4=)

Return Codes:

IBM Internal Use Only

Equate	Code (hex)	Description
	0000	Good return code
#BUSY	3000	RECEIVE port in use
#NOENTRY	3100	No port 0 for this program (not in system table)
#NOOCB	3200	OCB pointer missing from command
#NOTOPEN	3400	Program has not executed OPEN
#PORTRNG	3500	PORT number out of range
#RCVTIME	4000	Receive timeout expired
#TOOBIG	4100	Received message too big for buffer
#NNKEXIT	4200	No work exit taken

Note: If RECEIVE command without NOWORK parameter is issued and there are no messages waiting in the receive buffer, RECEIVE command will force the task issuing it to enter the wait queue until a message is received or (if specified) the receive time out expires. During this time, a task with a lower priority may become active.

IBM Internal Use Only

4.3.85 RESET

Function

RESET is used to reset or clear an event.

When an event occurs for which a task is waiting, the task will again become active. If the task were subsequently to issue another WAIT command for the same event, without taking any special action, the event is still defined as having "occurred" and no wait would be performed. It is necessary to define effectively the event as "not occurred" in order to cause a new wait to transpire. This is the function of the RESET command.

Events are named logical entities which are represented in storage by a system control block called an Event Control Block (ECB). Resetting an event is physically done by setting the first word of its ECB to '0'.

Syntax

label	RESET	event,KEY=,P1=,P2=
Required		event
Defaults		KEY=current partition
Indexable		event

Operands

event	The symbolic name of the event being reset. For an event associated with a macrofunction interrupt, use the name of the INTBIT command that defines the bit. (Do not confuse this reset operation with the resetting of the interrupt line on the macrofunction card. This interrupt must be physically reset by the application using the DISWRITE command). (P1=)
KEY=	The key value indicates which partition of memory contains the ECB being reset. This operand is described in "ATTACH" on page 36. (P2=)

IBM Internal Use Only

4.3.86 RETURN

Function

RETURN is used in a subroutine to provide linkage back to the calling program. A subroutine may have more than one RETURN command.

Syntax

label	RETURN
Required	None
Defaults	None
Indexable	None

Operands

None

4.3.87 SELECT

Function

SELECT defines a structured block. The SELECT block is a series of EDX statements delimited by WHEN statements. There can be a maximum of 255 WHEN statements associated with one SELECT statement. WHEN determines whether a statement is true or false. If the statement is true then control is passed to the true code, if it is false then control is passed to the next WHEN, OTHER or ENDSELECT statement.

Each SELECT block must have an associated ENDSELECT at the end of the block. Also, at least one WHEN and one OTHER statement is required.

Nested select blocks are allowed, maximum nesting level is 255.

Syntax

label	SELECT	(relational statement) ,SWITCH=
required	none	
default	none	
indexable	operand1, operand2	if relational statement is given.
	SWITCH	

Operands

statement	It is optional. If it is specified, the statement indicates the relationship to be tested. If the condition is true, execution proceeds sequentially. If the condition is false, a branch is made to the end of the select block.
SWITCH=	It is optional. If it is specified, it defines the variable to be tested by the WHEN statements enclosed in a given SELECT-ENDSELECT block.

The description of the syntax and the examples of relational statements can be found in "Program Sequencing" on page 26.

Examples:

1. SELECT with LEAVE and without SWITCH= or relational statement:

```

SELECT
  EDX statements
  WHEN (C,EQ,D,Z)
    EDX statements
    LEAVE
  WHEN (X,EQ,Y,FLOAT)
    EDX statements
    LEAVE
  WHEN (W,EQ,Z),OR,(X,EQ,Y)
    EDX statements
  .
  OTHER
  EDX statements
ENDSELECT
    
```

2. SELECT with relational statement only:

IBM Internal Use Only

```
SELECT (C,GT,0)
  EDX statements
  WHEN (C,EQ,D,2)
    EDX statements
  WHEN (X,EQ,Y,FLOAT)
    EDX statements
  WHEN (W,EQ,Z),OR,(X,EQ,Y)
    EDX statements
  .
  .
  OTHER
  EDX statements
ENDSELECT
```

3. SELECT with SWITCH only:

```
SELECT SWITCH=CASE
  EDX statements
  WHEN 1
    EDX statements
  WHEN 2
    EDX statements
  WHEN 3
    EDX statements
  .
  .
  OTHER
  EDX statements
ENDSELECT
```

4. SELECT with relational statement and SWITCH:

```
SELECT SWITCH=CASE,(CASE,GT,0)
  EDX statements
  WHEN 1
    EDX statements
  WHEN 2
    EDX statements
  WHEN 3
    EDX statements
  .
  .
  OTHER
  EDX statements
ENDSELECT

SELECT SWITCH=CASE,(CASE,GT,0)
  EDX statements
  WHEN 1
    EDX statements
  WHEN 2
    EDX statements
  WHEN 3
    EDX statements
  .
  .
  OTHER
  EDX statements
ENDSELECT
```

5. Nested SELECT statements:

IBM Internal Use Only

```

SELECT
  EDX statements
  WHEN (C,EQ,D,2)
    EDX statements
    SELECT SWITCH=CASE,(CASE,GT,0)
      EDX statements
      WHEN 1
        EDX statements
        SELECT SWITCH=CASE
          EDX statements
          WHEN 1
            EDX statements
          WHEN 2
            EDX statements
          WHEN 3
            EDX statements
          .
        OTHER
          EDX statements
      ENDSELECT
    WHEN 2
      EDX statements
    .
  OTHER
    EDX statements
  ENDSELECT
  LEAVE
  WHEN (X,EQ,Y,FLOAT)
    EDX statements
  LEAVE
  WHEN (W,EQ,Z),OR,(X,EQ,Y)
    EDX statements
  .
  .
  OTHER
    EDX statements
ENDSELECT

```

IBM Internal Use Only

4.3.88 SEND

Function

The SEND command transfers a message from one machine to a destination in another. Origin information is normally sent with the command, so a response can be returned if desired. If no response is needed, then 3 words of request data may be sent instead of origin information.

The user may either wait or specify an event to be posted when the transfer is completed. The return code is put in the SEND command and in the ECB code word (or code field of the TCB).

The SEND command MUST NOT be re-executed by any task until the transfer is complete.

Syntax

label	SEND	dest,buffer,count,EVENT=,PRIOR=,PORT=,DATA=, RC=,KEY=,P1=,P2=,P3=,P4=
Required		dest,buffer,count
Defaults		EVENT=,PRIOR=50,PORT=0
Indexable		KEY=current partition dest,buffer

Exclusions: PORT and DATA cannot both be specified

Operands

dest	The location of the destination control block. This may be a DEST definition or the origin field in a RECEIVE command. (P1=)
buffer	The location of the data to be transferred. (P2=)
count	The number of words to transfer. This may be a number between 0 and 500, or an assembler expression preceded by an '+'. (P3=)
EVENT=	The location of an ECB to be posted on completion of the transfer. If EVENT is not specified, the task will wait in the SEND command until the transfer is complete.
PRIOR=	Message priority; a number between 0 and 126, or an assembler expression preceded by an '+'. (P4=)
PORT=	The number of a receive port in the program to receive any response to this message. This will be put in the origin field of the message.
DATA=	The address of a 3 word block of request data to be sent instead of origin information.
RC=	A unique label to be attached to the return code field in the SEND.
KEY=	This parameter indicates which partition of DSC memory the buffer is located. (P4=)

Return Codes

Return codes from the SEND command are listed on the next page. Errors marked with an asterisk (*) are retried a preset number of times (currently 5) before an error code is returned to the user. If no response is received after a 'BLAST RESET', code x'8600' is returned regardless of the retry count. Thus, if the other side is not running, the SEND will be abandoned after only 3 transmissions.

A good return code means:

IBM Internal Use Only

- No transmission errors were detected.
- Destination was 'known'; either a loaded program or a 'loadable' service program in the SERIES/1 system table.
- Storage was available for the message; either a user's receive buffer or a system buffer.
- Message was successfully unloaded into the buffer.

Return Codes:

Equate	Code	Description
-----	----	-----
	0000	Good return code
	FFFF	Posted to SEND end event
#BUSY	3000	SEND command in use
#NOOCB	3200	OCB pointer missing from command
#NOTOPEN	3400	Program has not executed OPEN
#PORTRNG	3500	PORT number out of range
#COUNRNG	3600	Word count out of range
#PRIORNG	3700	Message priority out of range
#BADNODE	3800	Invalid destination node number
#NTFOUND	5000	Destination not found
#NTLOAD	5100	Destination marked 'NOT LOADABLE'
#TOOSMAL	5200	Receive buffer too small
#NOBUFF	5300	No free buffers available
#NORBUF	5500	No receive buffer supplied by RECEIVE command
#ACKTIME	8600 *	Timeout waiting for other side
#TXFAIL	8700 *	Transmit buffer not emptied
#ACKFAIL	8800 *	Receive error on acknowledgement from other side
#PARITY	E400 *	Parity error
#FRAME	E200 *	Framing error
#LRCERR	E900 *	LRC error
#BLAST	EB00 *	Blast reset received
#BADCNT	EC00 *	Invalid count field received
#DISRX	ED00 *	DIS error on RECEIVE
#INITBID	FF00 *	Initial bid from other side or other side timed out
#FRAGMNT	EF00 *	Fragment of Blast Reset received (not returned to user)
#BIDRESP	0100	Response to initial bid (not returned to user)
#DISTX	2200	DIS error on transmit (not returned to user)

* - re-triable errors

Note: If SEND command with EVENT= not specified is issued, a SEND command will force the task issuing it to enter the wait queue for the duration of the data transfer. During this time, a task with a lower priority may become active.

IBM Internal Use Only

4.3.89 SHIFTL

Function

Opnd1 is shifted left by the number of bit positions specified by opnd2. Vacated positions on the right are filled with zeroes. If opnd2 is a variable, it is assumed to be single-precision, and the shift count is its value modulo the precision of opnd1.

Syntax

label	SHIFTL	opnd1,opnd2,count,RESULT= P1=,P2=,P3=
Required		opnd1,opnd2
Defaults		count=1,RESULT=opnd1
Indexable		opnd1,opnd2,RESULT

Operands

opnd1	The name of the variable to which the operation applies; it cannot be a constant. If the RESULT= operand is not specified, then opnd1 is also the implicit result. (P1=)
opnd2	This operand determines the value by which the first operand is modified. Either the name of a variable or an explicit constant may be specified. (P2=)
count	The number of consecutive variables upon which the operation is to be performed. (P3=)
RESULT=	This operand may optionally be coded with the name of a variable or vector in which the result is to be placed. In this case, the variable specified by the first operand is not modified.

See "Data Movement, Logical and Arithmetic Functions" on page 18 for examples.

Note: When a task with a higher priority is waiting in the ready queue while a SHIFTL instruction with the count parameter greater than 255 is executing, a task switch will occur before the instruction is completed. The execution of the SHIFTL instruction will be resumed after the higher priority task terminates or enters a wait state.

IBM Internal Use Only

4.3.90 SHIFTR

Function

Opnd1 is shifted right by the number of bit positions specified by opnd2. Vacated positions on the left are filled with zeroes. If opnd2 is a variable it is assumed to be single-precision, and the shift count is its value modulo the precision of opnd1.

Syntax

label SHIFTR opnd1,opnd2,count,RESULT=
 P1=P2=P3=

Required	opnd1,opnd2
Defaults	count=1,RESULT=opnd1
Indexable	opnd1,opnd2,RESULT

Operands

opnd1	The name of the variable to which the operation applies; it cannot be a constant. If the RESULT= operand is not specified, then opnd1 is also the implicit result. (P1=)
opnd2	This operand determines the value by which the first operand is modified. Either the name of a variable or an explicit constant may be specified. (P2=)
count	The number of consecutive variables upon which the operation is to be performed. (P3=)
RESULT=	This operand may optionally be coded with the name of a variable or vector in which the result is to be placed. In this case, the variable specified by the first operand is not modified.

See "Data Movement, Logical and Arithmetic Functions" on page 18 for examples.

Note: When a task with a higher priority is waiting in the ready queue while a SHIFTR instruction with the count parameter greater than 255 is executing, a task switch will occur before the instruction is completed. The execution of the SHIFTR instruction will be resumed after the higher priority task terminates or enters a wait state.

4.3.91 SPACE

Function

The SPACE statement is used to insert one or more blank lines into the listing.

Syntax

SPACE value

Required	None
Defaults	value = 1

Operands

value

A decimal value specifying the number of blank lines to be inserted. If no value is entered, one blank line will be inserted. If this value exceeds the number of lines remaining on the page then an eject to the top of the next page is done.

For more information on the post-processor see "Listing Control Instructions" on page 32.

Note: The SPACE command has no effect while the post-processor is active. To perform the same function during post-processing, the *SPACE nn statement (starting in column 1) is used, where nn is a value between 1 and 99 and has a default of 1.

4.3.92 SQRT**Function**

Square root of opnd2 is placed in opnd1.

Syntax

label	SQRT	opnd1,opnd2,P1=,P2=
Required		opnd1,opnd2
Defaults		none
Indexable		opnd1,opnd2

Operands

opnd1	The name of a data area where the result is to be stored.
opnd2	The quantity on which the operation is to be performed. May be an address or an immediate value (constant or equate). If opnd2 is an address then the data at that address is assumed to be a signed floating point number of standard precision (double word). If opnd2 is immediate data it may be coded as an integer value between -32768 and 32767 and will be converted to a floating point number prior to performing the operation.

See "Derived Floating Point Functions" on page 22 for a more detailed explanation and examples.

After the operation is performed the return code is stored in the first word of the task control block (see "TCBGET" on page 161). Valid return codes include:

Return Code	Description
-----	-----
-1	Successful
1	Operand too large, or overflow in operation
3	Square root of a negative number
5	Operand too small, or underflow in operation
7	Hardware check. The APU failed to complete the execution of the command.

Note: The software registers (#1 and #2) may not be used as operands in floating point operations since they are only 16 bits in length. The registers may, of course, be used to specify the address of operands.

IBM Internal Use Only

4.3.93 STIMER

Function

STIMER is used to start a software timer and provide an interrupt after the specified number of milliseconds have elapsed. It allows a means of periodically executing a portion of the user program and/or providing accurate program delays. The minimum timer setting is 1 millisecond and the maximum setting is 60,000 milliseconds or 60 seconds.

STIMER may be used in conjunction with the WAIT command. Two STIMER commands without an intervening WAIT will cause the remainder of the time interval specified by the first STIMER to be replaced by the interval specified by the second STIMER.

Syntax

label STIMER count, WAIT, P1=

Required	count
Defaults	None
Indexable	None

Operands

count The address of a word, or an explicit constant, which specifies the timer setting in milliseconds. (P1=)

WAIT Specifies that control will not return until the time interval has elapsed. If WAIT is not specified, then a subsequent WAIT instruction may be issued with the keyword 'TIMER' specified as the 'event' being waited upon.

4.3.94 SUBROUT**Function**

SUBROUT is used to define the entry point of a subroutine. Up to five parameters may be specified as arguments in the subroutine. The subroutine must have a RETURN command to provide linkage back to the calling program. Nested subroutines are allowed, and a maximum of 999 subroutines are permitted per program. Recursive subroutines are not supported.

Syntax

```
label    SUBROUT  name,par1,...,par5
```

Required	name
Defaults	None
Indexable	None

Operands

name Name of the subroutine.

par1,... Names of arguments or parameters passed from the calling program. These names must be unique to the whole program. All parameters defined outside the subroutine are known within the subroutine. Thus, only parameters which may vary with each call to a subroutine need to be defined in the command.

For instance, assume two calls to the same subroutine. At the first call, parameters 'A' and 'C' are to be passed, while at the second, 'B' and 'C' are to be passed. Because 'C' is common to both, it need not be defined in the SUBROUT statement. However, a new parameter 'D' would be specified to account for passing either 'A' or 'B'.

```
CALL ROUTINE,A
CALL ROUTINE,B
.
.
.
SUBROUT  ROUTINE,D
.
```

Note: Do NOT code a subroutine within another subroutine. This will generate a wrong return address.

IBM Internal Use Only

4.3.95 SUBTRACT

Function

Signed subtraction of opnd2 from opnd1. May be abbreviated SUB.

Syntax

label SUBTRACT opnd1,opnd2,count,RESULT=,PREC=
P1=,P2=,P3=

Required	opnd1,opnd2
Defaults	count=1,RESULT=opnd1,PREC=S
Indexable	opnd1,opnd2,RESULT

Operands

opnd1	The name of the variable to which the operation applies; it cannot be a constant. If the RESULT= operand is not specified, then opnd1 is also the implicit result. (P1=)
opnd2	This operand determines the value by which the first operand is modified. Either the name of a variable or an explicit constant may be specified. (P2=)
count	The number of consecutive variables upon which the operation is to be performed. (P3=)
RESULT=	This operand may optionally be coded with the name of a variable or vector in which the result is to be placed. In this case, the variable specified by the first operand is not modified.
PREC=	This operand takes the form PREC=XYZ, where X applies to opnd1, Y to opnd2, and Z to the result. The value for each specification may be either S (single-precision) or D (double-precision). See "Mixed-precision Operations:" on page 19 for restrictions and abbreviations of the PREC= operand.

See "Data Movement, Logical and Arithmetic Functions" on page 18 for a more detailed explanation and examples.

IBM Internal Use Only

4.3.96 SYSCDEF

Function

The SYSCDEF statement is used to define a series of data structures and a displacement into the system common area (SYSCOM). These data structures will be created and moved to the designated part of SYSCOM by the SYSCOPEN command.

Syntax

```
label    SYSCDEF  displacement,(type,count,value/address),
                                     (type,count,value/address),
                                     (type,count,value/address)
```

Required	displacement,type,count,value/address
Defaults	count=1,value=0
Indexable	None

Operands

displacement	An equated value (+ or -) for specifying where in SYSCOM the defined data structures are to be moved by the SYSCOPEN command.
type	Defines the type of data structure which includes: ECB, QCB, and DATA. The data types can be multiply defined and in any order.
count	Defines the number of data structures to be created in a sequence, all with the same type, value / address. A minimum of 1 and a maximum of 65768 replications are allowed.
value / address	Defines the initialization value or address to be set into the data structure. The value or character string defined must fit in a 1 word (16 bit) field.

Further Definition of Operand 2:

1. type - 'ECB' 'QCB' 'DATA'
2. count - integer
3. value - signed decimal
- signed decimal
- hexadecimal
- character (2max)
4. address - A(label)

For example, the following code segment:

```
#FIVE      EQU      5
KOM        DATA    F'0'
.
.
SYS1       SYSCDEF  #FIVE,(ECB,1,-1),      one ecb
                                     (QCB,2,-2),    two qcb's
                                     (ECB,2,5),    two ecb's
                                     (DATA,3,F'8'),  3 words of '8'
                                     (DATA,4,x'9999'), 4 words of x'9999'
                                     (DATA,5,c' '), 5 words of blanks
                                     (DATA,1,A(KOM)) 1 word address of KOM
                                     .
                                     .
                                     .
                                SYSCOPEN SYS1
```

would format the system common area as follows (ECB's occupy 3 words and QCB's occupy five words):

IBM Internal Use Only

ADDRESS:	CONTENTS:
-----	-----
SYSOM + 5	ECB -1
+ 8	QCB -2
+ 13	QCB -2
+ 18	ECB 5
+ 21	ECB 5
+ 24	DATA F'8'
+ 25	DATA F'8'
+ 26	DATA F'8'
+ 27	DATA X'9999'
+ 28	DATA X'9999'
+ 29	DATA X'9999'
+ 30	DATA X'9999'
+ 31	DATA C' '
+ 32	DATA C' '
+ 33	DATA C' '
+ 34	DATA C' '
+ 35	DATA C' '
+ 36	DATA A(KOM)

4.3.97 SYSCOPEN**Function**

The SYSCOPEN command allows a program to format a part of the system common area (SYSCOM) with constants, variables, addresses, ECB's and QCB's. The SYSCDEF statement, specified in the SYSCOPEN command, defines the data structures are where the data structures will be located in SYSCOM. SYSCOM must be formatted using SYSCOPEN and SYSCDEF because the ECB's and QCB's contain validation fields which are aligned with memory. The supervisor verifies ECB's and QCB's by checking the validation fields. The SYSCOPEN command creates the correct validation value when it moves these structures into SYSCOM.

Syntax

label SYSCOPEN syscdef

Required	syscdef
Default	none
Indexable	None

Operands

syscdef

The label on a SYSCDEF statement. The SYSCDEF statement contains a list of data structures, their initial values and where these data structures are to be loaded into SYSCOM.

IBM Internal Use Only

4.3.98 TASK

Function

The TASK statement defines the beginning of a block of code which will execute asynchronously with other tasks in the system according to its assigned priority.

Each task in a program, except the 'main task', begins with a TASK statement and must be terminated with an ENDTASK. The main task begins with the PROGRAM statement and is terminated by ENDPORG.

Syntax

taskname TASK start,priority,EVENT=,TERMERR=

Required	taskname,start
Defaults	priority=150
Indexable	None

Operands

taskname	The name of the task. A system control block known as the Task Control Block or 'TCB' is generated for each task in an application program. For further information see the description of the 'taskname' operand of "PROGRAM" on page 131.
start	The label or address of the first instruction executed when the task is first attached.
priority	The priority to be assigned to the task. The range is from 0 (highest priority) to 255 (lowest priority). Priorities separate tasks according to the relative criticality of their real time needs for CPU time. Priority assignments must therefore account for other programs expected to be executing simultaneously and should be made in consultation with other users.
EVENT=	This event (END ECB) of the task will be posted at the termination (detaching) of this task. The attaching task can, if desired, synchronize its operation by issuing a WAIT for this event.
TERMERR=	Label of the user routine that is to be given control if an unrecoverable I/O error occurs.

IBM Internal Use Only

4.3.99 TCBGET

Function

The TCBGET instruction obtains data from a specified field in the task control block (TCB) of the currently executing task.

Syntax

label TCBGET opnd1,opnd2,P1=

Required: opnd1

Defaults: opnd2 returns address of current TCB

Indexable: opnd1

Operands

opnd1

The label of a one word data area where the system stores the TCB field specified by opnd2. (P1=)

opnd2

The name of the TCB field to copy. If this operand is not coded, the address to the current TCB will be returned.

This operand can be coded using any of the TCB equate names. Some of the more useful fields in the TCB are:

#TCBC0 The first word of the TCB. Many instructions place return codes here.

#TCBC02 The second word of the TCB.

#TCBADS The current address key.

Examples:

TCBGET	TCBADDR	Put address of current TCB in TCBADDR
TCBGET	#1	Put address of current TCB in #1
TCBGET	RC,#TCBC0	Put first word of TCB in RC
RC	DATA F'0'	
TCBADDR	DATA F'0'	

IBM Internal Use Only

4.3.100 TEXT

Function

TEXT is used to define a standard text message. The characters are stored in EBCDIC code. The PRINTTEXT command or any I/O instruction that has a prompt message capability may be used to print this message on the console.

Syntax

label	TEXT	'message',LENGTH=
Required		'message' or LENGTH=
Defaults		None
Indexable		None

Operands

'message'	Any text string defined between apostrophes. If this operand is not coded, LENGTH= must be coded. Use a pair of apostrophes to represent each printable apostrophe. The symbol '@' will cause a skip to the next line unless a PRINTTEXT instruction is used with MODE=LINE.
LENGTH=	Defines the maximum size (in words) of the text buffer. If this operand is not coded, 'message' must be coded. Each word contains 2 characters. Maximum value = 127 words. If both 'message' and LENGTH= are coded, any remaining positions will be initialized with blanks (X'40').

Examples:

MSG1	TEXT	'A MESSAGE'
MSG2	TEXT	'ABC',LENGTH=10
MSG#	TEXT	LENGTH=20

IBM Internal Use Only

4.3.101 TIME

Function

TIME is used to set the internal time of day clock and the date from a specified location.

Syntax

label	TIME	loc,P1=
Required		loc
Default		None
Indexable		None

Operands

loc

The address of the 6 word block containing the time and date (in binary) used to set the system clock. The time and date must be formatted as follows: Hour, Minutes, Second, Month, Day, Year. (P1=)

Example:

```
TIME1 DATA F'12' .hour
      DATA F'13' .minute
      DATA F'14' .second
      DATA F'1' .month
      DATA F'2' .day
      DATA '86' .year
      .
      .
      TIME TIME1 .set time to 12:13:14 & date to 1/2/86
```

Note: The nucleus does not check the time or date supplied to make sure it is valid.

4.3.102 TITLE

Function

The TITLE instruction allows the user to define a title to be placed at the top of subsequent pages.

Syntax

label	TITLE	'message'
Required		message
Defaults		None

Operands

message	An alphanumeric character string, enclosed in apostrophes, up to 100 characters in length.
---------	--

A program may contain more than one TITLE statement. Each one causes following pages to be printed with the new title at the top. This heading is repeated on each new page until a new TITLE statement is encountered. The TITLE statement causes an EJECT to the next page.

IBM Internal Use Only

4.3.103 USER

Function

USER creates linkage to a 'user exit routine'. This provides the user a means of programming (in JIB' assembler language) a function which is not supported by the EDX/J instruction set. The user exit routine must follow certain conventions as described below.

Note: If USER commands are to be included in the EDX/J program, the program must be preceded by the following statement in order to invoke the microcode assembler:

COPY JIBPMAC

Syntax

label	USER	name, PARM=(parml, ..., parmn), P=(namel, ..., namen)
Required		name
Defaults		None
Indexable		None

Operands

name	The entry point name of the user exit routine (JIB' instruction stream)
PARM=	A list of parameters that are to be passed to the user routine. Each parameter may be either a single precision constant or an address label.
P=	A list of names to be attached to the PARM operands.

Conventions

Upon entry into the user exit routine, the following registers are significant:

#R4, #R5 address to be returned to at end of the routine

#R6, #R7 address of caller's task control block (TCB)

#R10, #R11 address of first parameter in parameter list of the USER command

At the end of the routine all of the registers mentioned above must contain the same value as contained upon entry to the routine. The routine must end with the following instruction:

BALR #R4, #R4

The user routine may access parameters passed to it by referencing register pair #R10, #R11. For example, to load the second parameter into register pair #R14, #R15, the following instructions would be used:

```

:
:
AIP  #R10,1
LD   #R14,#R10
:
:
```

In this example, register pair #R10, #R11 may be restored to its original value prior to returning to the nucleus by using the following instruction:

```

:
:
SIP  #R10,1
:
:
```

IBM Internal Use Only

The 'user exit routine' is free to use all registers provided that the registers mentioned above (#R4, #R5, #R6, #R7, #R10, #R11) are restored to their original value before leaving the user exit routine.

The 'user exit routine' must be coded as a subroutine outside of the EDX/J instructions stream.

Example:

```
IREGSTR DATA F'0'          .contents of I register
      .
      USER READI,PARM=(IREGSTR) .call user routine
      .
*
*      user routine to read contents of I register
*
READI  LDR  #R8,#R10          .#R8 <- addr of first parm
      LD   #R8,#R8           .#R8 <- addr of IREGSTR
      LI   #R0,0             .#R0 <- 0
      MRX  #R1,IREG          .#R1 <- contents of I register
      ST   #R0,#R8           .IREGSTR <- contents of I register
      BALR #R4,#R4           .return to EDX/J
```

Note:

1. The following JIB' instructions should NOT be used because some operands used in them are not relocatable:
B , BAL , BALL , BALLR , LAD , LAH , LAL .
2. All conditional branch instructions may be used since application programs are loaded on 256 word boundaries.
3. The JIB' instruction 'RETURN' is not available due to its name duplication with the EDX/J 'RETURN' command.

IBM Internal Use Only

4.3.104 WAIT

Function

The WAIT command is used to wait for the occurrence of an event such as the completion of an I/O operation or a macrofunction interrupt. Each event has an associated name specified by the user. The status of any event defined by the user is 'event occurred' unless the user explicitly RESETs the event name to zero before issuing the WAIT or by resetting the event in the WAIT command as described below.

Syntax

label	WAIT	event,RESET,TIMEOUT=,KEY=,P1=,P2=,P3=
Required		event
Defaults		event not cleared before wait
Indexable		KEY=current partition
		event

Operands

event	The symbolic name of the event being waited upon. (P1=)
	For a macrofunction interrupt, use INTx, where 'x' is the interrupt bit identifier as assigned in the INTBIT command.
	For time intervals set by STIMER, use TIMER as the event name. Do not code RESET or TIMEOUT= with TIMER.
RESET	Reset or clear the event before waiting. Using RESET will force the wait to occur even if the event has occurred or been posted previously. (Do not confuse this reset operation with the resetting of the interrupt line on the macrofunction card. This interrupt must be physically reset by the application using the DISWRITE command.)
TIMEOUT=	Specify the number of milliseconds (1-60,000) to be used as a maximum wait time. If the event has not occurred within the time specified, the event will be posted with a post code of X'FFFE' (decimal -2).
	Note: The timeout option should not be used when more than one task may wait on the same event. Under these circumstances, the results may be erroneous. (P2=)
KEY=	The key value indicates which partition of memory contains the ECB that describes the event being waited for. This operand is described in the ATTACH instruction. (P3=)

Note: The WAIT instruction is NOT to be used within an ATTNLIST routine.

IBM Internal Use Only

4.3.105 WHEN

Function

WHEN determines whether a statement is true or false. If the statement is true, the true code is executed, and then the control is passed outside the SELECT block; the first statement following the ENDSELECT statement is executed. If the statement is false, the control is passed to the next WHEN, OTHER or ENDSELECT statement.

There can be a maximum of 255 WHEN statements associated with one SELECT statement.

Syntax

```
WHEN (statement) ,AND | OR,(statement)
required      none
default       none
indexable     operands in each statement.
```

Operands

```
statement
```

The statement indicates the relationship to be tested. If SWITCH= is specified, the statement to be tested must be a single operand.

Examples:

```
WHEN (XYZ,EQ,ABC)
WHEN CASE1
WHEN 22
```

IBM Internal Use Only

4.3.106 WHEREs

Function

Locates another program executing elsewhere in the system. WHEREs searches each partition in ascending order from the nucleus partition (FF) through partition n to determine if the program is contained in that partition. It indicates results of that search by placing a return code in the first word of the task control block. If more than one copy of the program exists, the system reports only the first copy found.

Syntax

label	WHEREs	programe, address, KEY=, P1=, P2=, P3=
Required		programe
Defaults		None
Indexable		None

Operands

programe	The label of an eight byte area containing the 1-8 character program name of the program to be located. If the label has fewer than eight characters, the program name must be left justified and padded with blanks on the right. The program name must begin on a full word boundary. (P1=)
address	The label of a word in which the load point address of the located program will be returned if the program is found. This address is the first byte of the program and is also the beginning of the program header. If the program is not located, a -1 is stored at this location. (P2=)
KEY=	The label of a word in which the address key of the partition containing the located program will be returned if the program is found. The address key is one less than the partition number. The HAZEL nucleus partition is denoted by the X'FF' partition number. (P3=)

Return Codes

Return codes are returned in the first word of the task control block (TCB) of the program or task issuing the instruction. The label of the TCB is the label of your program or task (taskname).

Return Code	Description

FFFF	Program found
0000	Program not found

Examples:

WHEREs	PROGRAME, ADDRESS, KEY=TKEY
WHEREs	PROGRAME, ADDRESS
WHEREs	PROGRAME, KEY=TKEY
WHEREs	PROGRAME
WHEREs	*, *, *, P1=PROGRAME, P2=ADDRESS, P3=TKEY
WHEREs	,,, P1=PROGRAME, P2=ADDRESS, P3=TKEY
WHEREs	P1=PROGRAME, P2=ADDRESS, P3=TKEY
PROGRAME	DATA 8C' '
or	
PROGRAME	TEXT LENGTH=4

IBM Internal Use Only

4.3.107 WRITE

Function

The WRITE instruction transfers one or more records from a buffer area to a disk or diskette you define. Records are defined as being 256 bytes long and the buffer size should be in multiples of records. Successive sequential access to a dataset is not supported internally and must be supported by the application program by modification of the 'relrecno' parameter. Cross-partition transfers are supported by the WRITE command through the KEY= parameter.

Syntax

label	WRITE	dscb,loc,recnt,relrec,KEY=,WAIT=,END=,ERROR=,PREC=, P1=,P2=,P3=,P4=,P5=
Required		dscb,loc
Defaults		recnt=1,relrec=1
Indexable		loc,count,relrec

Operands

dscb	The label of a DSCB which contains the dataset's 'name' and possibly the volume id for that dataset. See "DSCB" on page 65.
loc	The label of the buffer area from which the data is to be transferred. This buffer need not be in the same partition as the Write command. see KEY=.
recnt	The number of contiguous records to be written. If the program sets the field to 0, no I/O operation is performed. A count of the actual number of records transferred is set in the second word of the task control block. If an end of dataset condition occurs (writing beyond the dataset limits), the system writes as many records as will fit the space remaining in the dataset and returns and end of dataset return code to the program.
relrec	The relative record number within a dataset where a write operation will start. This parameter can be formatted as a self-defining positive decimal value (or equate) or as a label of a variable containing the self-defining value. Note, relative record numbers must start from one (not zero). The S/1 uses the relrec=0 to indicate a sequential disk operation; this is currently not supported in the DSC. The POINT command which controls sequential disk operations is currently not supported in the DSC.
	An EOD indication is returned if an attempt is made to access a record outside or beyond the physical dataset.
	If you code a self-defining term, or an equated value indicated by a plus sign (+), then it is assumed to be a single-word value and is, therefore, generated as an inline operand. Because this is a one word value, it is limited to a range of 1 to 32767 (x'7FFF').
	If you code an indexable value or an address for this operand, the PREC operand can be used to further define whether relrecno is to be a single or double word value. See PREC=.
PREC=	This operand further defines the relrec operand when you code an address or an indexable value for that operand. PREC=S (the default) limits the value of relrec to a single precision value (maximum 32767).
END=	The label of the first instruction of the routine to be invoked if an end of dataset condition is detected during the WRITE operation (return code=10). If you do not code this

IBM Internal Use Only

ERROR= operand, the system treats an end of dataset condition as an error. Do not code this operand if you code WAIT=NO.

The label of the first instruction of the routine to be invoked if an error condition occurs during the execution of this operation. If you do not specify this operand, control passes to the instruction following the READ and you must test the return code in the first word of the task control block for errors. Do not code this operand if you code WAIT=NO.

WAIT= YES (default), to suspend the current task until the operation is complete. NO, to return control to the current task after the operation is initiated. Your program must issue a subsequent WAIT on the label of the DSCB used in the WRITE command to determine when the operation is complete. You can not code the END and ERROR operands if you code WAIT=NO. You must subsequently test the return code in the Event Control Block (ECB) named in the DSCB or in the first word of the task control block (TCB). See "TCBGET" on page 161.

Two return code are significant; -1 indicates a successful end of operation and +10 indicates 'end_of_dataset' which may be of logical significance and not just an error. For programming purposes, any other return codes should be treated as errors.

Px= The Px parameters are used to append label to other parameter in the WRITE command. The user can then modify these variables using those labels to control the WRITE operation. Note: the parameters must be modified in the same sense (ie. value or address) consistent with their original parameter definition. Px ASSIGNMENTS:

P1 - dscb	address
P2 - loc	address
P3 - recnt	value / address
P4 - relrec	value / address
P5 - KEY=	value / address

Note: The WRITE instruction without WAIT=NO parameter specified, will force the task issuing it to enter the wait queue for the duration of the data transfer. During this time, a task with a lower priority may become active.

IBM Internal Use Only

IBM Internal Use Only

5.0 SUPERVISOR UTILITY FUNCTIONS

5.1 NATIVE SUPPORT

The following Supervisor Utility Functions are activated by depressing the ATTENTION key (PA1 on 3270 keyboard, PF7 on 3101) and entering the appropriate commands in response to the prompting message:

\$?

displays the following:

- release level of the supervisor
- assembly date of the nucleus
- a list of the supervisor utility functions

\$A key

displays the names and load points of active programs in any one or all partitions of storage. The **KEY** parameter is optional.

\$BP

brings the system up ready to be used with no predetermined load operation or prompt; for example loading TCINT. This function can only be used on a production type system with 3101 terminals. **\$BP** is available only during a 10 second period immediately after the nucleus is loaded.

\$C (program_name load_point key | program_name |)

cancels a program and frees the storage that it occupied. The program name and key are to be entered as prompted. When more than one copy of a program is in main storage, the user will be prompted to specify the load point of the one he wishes to cancel.

\$CP key

replaces the currently active partition with the one specified.

\$D (addr word_count key base_addr | addr |)

displays the contents of storage at a user specified address. The starting address, as well as the key and the number of locations to be displayed, may be entered on the command line or in response to the prompt.

\$DIS

invokes a utility to execute the following Internal DIS Commands:

AD - Add interrupt bit to the poll list (including local block). Note, that if the user specifies a bit in a block which is not yet in the poll list, the following operations are done for that block:

- pending interrupts are reset
- interrupts are enabled.

During these operations a DIS error may occur, in which case poll list is not updated.

BP - Begin polling LSAs specified in the poll list.

CL - Card List. Example:

```
CARD ID = 0030
-----
BLOCK = 0000  MACRO = 0005
BLOCK = 0001  MACRO = 0001
BLOCK = 0001  MACRO = 0004
BLOCK = 0002  MACRO = 0006
BLOCK = 0012  MACRO = 0007
BLOCK = 001A  MACRO = 0006
```

IBM Internal Use Only

Note:

1. The CARD ID is the macrofunction card id in hex.
2. The BLOCK is the block number (in hex 0 - 1F) where the macrofunction card was located.
3. The LSA is the LSA at which the macrofunction card was located.

DE - Delete interrupt bit from the poll list (including local block).
Note, that polling is automatically suspended when the last block is deleted from the poll list.

FC - Find Card. Example:

CARD LIST BLOCK = 0000

```

MACRO = 0000 CARD ID = AP/DIS CARD
MACRO = 0007 CARD ID = AP/CSS CARD
MACRO = 0002 CARD ID = ARIEL CARD
MACRO = 0000 CARD ID = 0001
MACRO = 0004 CARD ID = 001E
MACRO = 0005 CARD ID = 002C
  
```

Note:

1. The title indicates that the following cards were found in block 0.
2. The LSA is the LSA at which the macrofunction/processor card was located.
3. The CARD ID is the macrofunction card id (in hex) or the name of a processor card.

EN - Exit the \$DIS utility.

RE - Read DIS Error Log. Example:

LSA NUMBER	-----ERRORS----- DIS ERRORS	TIME-OUTS
0000	5	9
001F	255	33
0022	0	1
003E	12	0

Note:

1. The ERRORS are divided into TIME-OUTS and DIS ERRORS. See "DISIO - ARIEL" on page 53 for DIS error description.
2. The maximum error count is 255. If the error count reaches 255 it is not reset and it does not wrap back to 0; it can be reset using the RS command.

RP - Read content of the poll list (including local block). Example:

BLOCK NUMBER	INTERRUPT	BITS	-----ERRORS----- DIS ERRORS	TIME-OUTS
0000	0	0 0 1 0 0 1 0	5	9
0001	1	1 1 0 0 0 0 0	255	33
0003	0	0 0 1 0 0 0 0	0	1
001A	1	0 0 0 0 0 0 1	12	0

Note:

1. The BLOCK NUMBER is the LSA of the BIC.
2. The INTERRUPT BITS marked (1) indicate the interrupt bits which are known to the application programs.
3. The ERRORS are divided into TIME-OUTS and DIS ERRORS. See "DISIO - ARIEL" on page 53 for DIS error description.
4. The maximum error count is 255. After the error count reaches 255 it is no longer incremented. The user may use the RS command to reset the error count back to zero.

RS - Reset DIS Error Log.

IBM Internal Use Only

SP - Suspend polling. After this command is issued, Hazel will not be able to receive interrupts from the remote blocks. Note, that interrupts from the local block are not affected by this command.

\$L (program_name key | program_name |)

loads a program from the disk to a partition of the DSC storage and starts it.

\$LH (program_name key | program_name |)

loads a program from the host to a partition of the DSC storage, and starts it. The program name along with the key (optional) is entered as prompted. The program name must be five characters or less. The host Series/1 will prefix the name you supply with "Ixx" (where xx is the tool controller line number) to form the name of the program on the Series/1 disk. If "Ixx program name" doesn't exist the Series/1 will attempt to send "T99 program name". If both "Ixx program name" and "T99 program name" exist on the Series/1 then "Ixx program name" will be sent.

\$LNG (program_name key | program_name |)

loads a program from the disk to storage as does \$L but does not start it.

\$LHNG (program_name key | program_name |)

loads a program from the host to storage as does \$LH but does not start it.

\$P (addr key base_addr | addr |)

allows to patch memory.

\$S (program_name load_point key | program_name |)

starts (ATTACHes) a program which has already been loaded. The program name and key are to be entered as prompted.

\$T

sets the system date and time.

\$W

displays the system date and time.

\$X

inserts a breakpoint at the prompted address specified by the user. Note, that this function is not available on systems with 3101 terminals. An equivalent function exists on systems with 3101 terminals; see "Stop EDX" on page 195.

Note:

1. If the ATTENTION key is pressed while the terminal is enqueued (or waiting for input), the Supervisor Utility Function will gain control of the terminal only after the current controlling task has released it. However, the supervisor does 'remember' that the ATTENTION key has been pressed.
2. When using the Utility functions that accesses user memory, a partition value is usually required along with other parameters. If no value is given then the partition default is dependant on the function being used as summarized in the following table.

IBM Internal Use Only

<u>Function</u>	<u>Default</u>
\$A	all partitions
\$C	partition that contains the program
\$D	current partition
\$LH,\$L	partition that will best contain the program
\$LHNG,\$LNG	partition that will best contain the program
\$P	current partition
\$S	current partition

5.2 REMOTE SUPPORT

A subset of the Hazel Supervisor Utility Functions may be executed remotely from S/1. The name of the S/1 program required for the remote support is SLDEBUB. The SLDEBUB program provides the following functions:

TC - SELECT TOOL CONTROLLER	EN - END DEBUG
QU - QUIESCE DEBUG	? - DISPLAY MENU
\$A - LIST ACTIVE PROGRAMS	\$D - DUMP MEMORY
\$C - CANCEL DSC PROGRAM	\$P - PATCH MEMORY
PD - PROGRAM TCB DUMP	DP - DUMP TO PRINTER
LH - LOAD A DSC PROGRAM	LN - LOAD A DSC NUCLEUS

IBM Internal Use Only

6.0 HAZEL/3 AND HAZEL/4 - MINI DEBUG OVERVIEW

The Hazel/3 and Hazel/4 (without AP) debug tool (Mini Debug) is available as part of the machine check handler code. This tool provides the user with a small set of functions which will assist in development and debugging of both EDX and JIB code.

Once the DSC has been IPLed, the user may access Mini Debug at any time by hitting the TEST REQ key.

Upon initial entry to Mini Debug, the user is presented with a register dump screen (see Figure 4 on page 179 and Figure 3 on page 179). Mini Debug provides the following functions:

- Display LSRs and PSWs
- Display LSR chapter and extended PSWs
- Display and patch memory
- Start/stop EDX Trace
- EDX single step
- JIB single step
- Start/Abort memory dump to S/1 diskette

The following is a summary of the function keys:

TEST REQ - ENTER MINI DEBUG AND DISPLAY REGISTER DUMP

PA2 - EXIT MINI DEBUG (from both displays)

PF1 - RETURN TO REGISTER DUMP (from memory display)

PF2 - TURN EDX TRACE ON/OFF (from register display)

PF3 - REPEAT JIB SINGLE STEP (from memory display)

PF5 - DISPLAY LSR CHAPTER AND EXTENDED PSWs (from register display)

PF6 - CHANGE MEMORY PARTITION (from memory display)

PF7 - SET/REMOVE EDX BREAKPOINT (from memory display)

PF8 - START MEMORY DUMP TO SERIES/1 DISKETTE (from register display)

PF9 - ABORT MEMORY DUMP TO SERIES/1 DISKETTE (from register display)

PF10 - SET/REMOVE JIB BREAKPOINT (from memory display)

PF11 - SET/REMOVE JIB BREAKPOINT (from memory display)

PF12 - SET/REMOVE JIB BREAKPOINT (from memory display)

IBM Internal Use Only

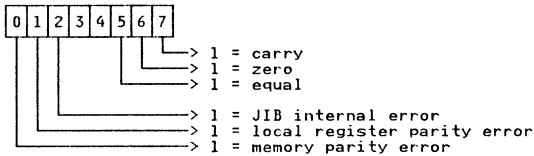
6.1 DISPLAY LSRS AND PSWS - PF1 OR TEST REQ

To enter register dump hit TEST REQ key, if not already in Mini Debug, or PF1 key, if in Mini Debug memory display. Figure 3 on page 179 and Figure 4 on page 179 show register dump screens for Hazel/4 and Hazel/3.

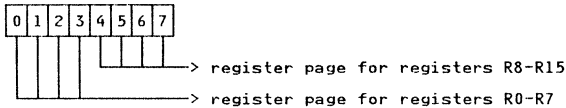
6.1.1 PSWs

There are eight pairs of the Program Status Words (PSWs). Each pair represents PSW of one level:

- the first word in each pair is the current instruction address register (IAR)
- the second word consists of two bytes:
 - the first byte contains the JIB errors and condition codes:



- the second byte shows what LSPRs were assigned to a given level.

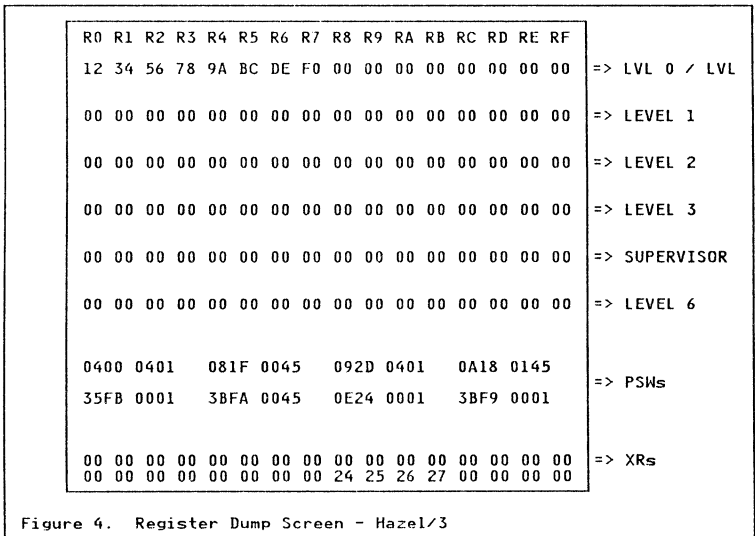
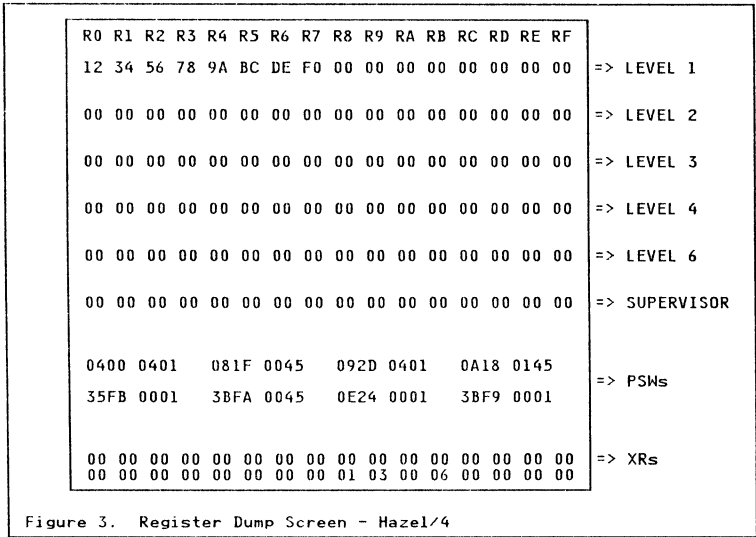


In the first row the PSWs of level 0 to 3 are displayed and in the second row PSWs of level 4 to 7.

6.1.2 XRs

There are 32 external registers (XRs). In the first row XRs 00 to 0F and in the second row XRs 10 to 1F. The explanation of the meaning of the XRs can be found in the Hazel nucleus listing.

IBM Internal Use Only



IBM Internal Use Only

6.2 DISPLAY AND PATCH MEMORY

After entering Mini Debug via TEST REQ key, the user must hit ENTER key to display/patch memory. Three functions are available:

1. view memory
2. patch memory
3. change partition (in Hazel/4 only)

6.2.1 Viewing Memory

The user may use two methods to view memory:

1. specifying memory address
2. scrolling

6.2.1.1 Specifying Memory Address

The user may enter hexadecimal memory address. The address will appear in the upper right corner of the screen. Pressing ENTER key will display memory at the specified address.

6.2.1.2 Scrolling

The user may scroll memory using the ARROW keys:

- UP ARROW - causes memory address to be incremented by X'0080'.
- DOWN ARROW - causes memory address to be decremented by X'0080'.
- LEFT ARROW - causes memory address to be incremented by one.
- RIGHT ARROW - causes memory address to be decremented by one.

6.2.2 Patching Memory

Once in memory display screen, to patch memory the following sequence of operations is required:

1. select address of the word to be patched
2. press INSERT MODE key
3. P=XXXX (XXXX is the address to be patched) will appear in the upper left corner.
4. enter the patch
5. press SPACE bar to save the last word patched and then patch the next word
6. press ENTER to save the last word patched and then exit

IBM Internal Use Only

6.3 OTHER MINI DEBUG FUNCTIONS

6.3.1 Start/Stop EDX Trace - PF2

Note, that PF2 is active only in register display screen. PF2 key is used to start/stop the EDX trace.

There are two trace buffers:

1. small buffer - 256 words starting at label \$BRINGUP
2. large buffer - X'A000' to X'FF00' (in partition 2 for Hazel/4 and up)

The small trace buffer is the default. To select the large trace buffer, the user must patch the EDX trace pointer to X'A000'. The address of the EDX trace pointer can be located via the Nucleus Directory; see Figure 56 on page 293 and Figure 57 on page 294 under 'CURRENT TRACE ADDRESS'.

The EDX trace has four types of entries:

EDX TASK - shows what EDX instructions were executed by a given task.

```
A000: 5AB2 5322 5328 532F 5338 5342 534A 5354
A008: 5AB2 535C 5362 5368 0000 0000 0000 0000
```

In the example above, the TCB address (5AB2 at location A000) is followed by the addresses of EDX instructions (5322 5328 ... starting at location A001). Note, that:

- each entry is eight words long and starts with the TCB address
- 0000 are trailers and they indicate that next line in the EDX trace contains a POST entry, a CHAIN entry or a new task entry.

POST - example:

```
A000: 5AF4 5B23 5B28 5B2F 0000 0000 0000 0000
A008: FF04 4730 0004 47E8 0000 0000 0000 0000
```

The first word of the POST entry consists of the POST marker (FF), the level (04) from which post was executed. The second word is the address (4370) of the posted ECB.

CHAIN - example:

```
A000: 5AF4 5B23 5B28 5B2F 0000 0000 0000 0000
A008: FF04 4730 0004 47E8 0000 0000 0000 0000
```

The first word of the CHAIN entry consists of the CHAIN marker (00), the level (04) from which chain was executed. The second word is the address (47E8) of the TCB chained to the Ready Queue.

END - shows the last address of the last EDX instruction traced.

```
A000: 0006 5CE8 0000 0000 0000 0000 0000 0000
A008: 5CE8 5B0F 5B14 FFFF FFFF FFFF FFFF FFFF
```

In the example above, the last EDX instruction traced was at address 5B14. The END entry consists of FFFF characters.

6.3.2 Repeat Stop JIB - PF3

Note, that PF3 is active only in register memory screen. The PF3 key may be used when JIB is already stopped on a given address and the user wishes to stop JIB again on the same address. This function may be used if for example, one wishes to trace JIB execution in a loop.

IBM Internal Use Only

The PF3 key performs the following sequence of operations:

1. new breakpoint is set at the current address incremented by one
2. the previous breakpoint is removed
3. JIB resumes execution and stops at the new breakpoint
4. new breakpoint is set at the current address decremented by one
5. the previous breakpoint is removed
6. JIB resumes execution and stops at the new breakpoint

When the breakpoint address is reached the process check light will start flashing.

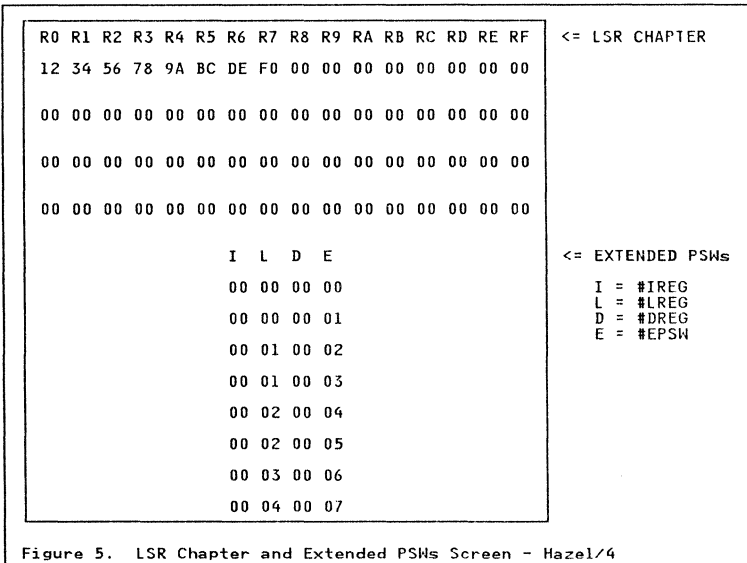
To verify that the stop occurred where the breakpoint was set, the user should check PSWs in the register dump. One of the PSWs should be equal to the breakpoint set by the user.

6.3.3 Display LSR Chapter and Extended PSWs - PF5

Note, that PF5 is active only in register display screen. To display LSR chapter and extended PSWs the following sequence of steps is required:

1. while in register display screen press PF5 key
2. enter LSR chapter 0 to 7

The figure below shows the 'LSR Chapter and Extended PSWs' screen.



6.3.4 Change Partition (Hazel/4) - PF6

Hazel/4 memory is divided into four partitions. Addresses X'0000' to X'7FFF' may be accessed from all four partitions. Addresses X'8000' to

IBM Internal Use Only

X'FFFF' require partition number for full qualification. The default partition is partition 1.

To change the partition number the following sequence of steps is required:

1. press PF6 key
2. enter partition number 1 to 2 (note, that no prompt will appear)

Note, that PF6 is active only in memory display screen.

6.3.5 EDX Stop on Address - PF7

Note, that PF7 is active only in register display screen. To set a breakpoint for the EDX stop on address function the following steps are required:

1. while in memory enter EDX stop address
2. hit ENTER key
3. hit PF7 key
 - EDX stop address will be set for the address appearing in the left top corner of the screen
 - the EDX instruction will be set to FFFF
4. hit PA2 to exit Mini Debug
5. wait for machine check
6. when machine check occurs hit PF1 to enter Mini Debug (if machine check does not occur, the stop address was not executed)

To remove a previously set breakpoint the user must hit PF7 key while in memory display screen. This will cause the EDX instruction at the breakpoint address to be restored. After the breakpoint is removed EDX will resume normal execution.

When the breakpoint address is reached the process check light will start flashing.

To verify that the stop occurred where the breakpoint was set, the user should check LSR pair R1011 in the register dump. The R1011 pair on the level on which the EDX interpreter is executing contains the address of the currently executed EDX instruction.

6.3.6 Memory Dump to S/1 Diskette - PF8/PF9

Note, that PF8 and PF9 are active only in register display screen. PF8/PF9 key are used to start/stop memory dump to S/1 diskette. See "Appendix G. Memory Dump to S/1 Diskette" on page 319 for detailed description of the dump procedure.

6.3.7 JIB Stop on Address - PF10, PF11 or PF12

Note, that PF10-12 are active only in register display screen. To set a breakpoint for the JIB stop on Address function the following steps are required:

1. while in memory enter JIB stop address
2. hit enter key
3. hit PF10 key
 - JIB stop address will be set for the address appearing in the left top corner of the screen
 - the JIB instruction will be set to 0XXX (XXX is such that 0XXX is a "branch to itself" instruction.)
4. hit PA2 to exit Mini Debug
5. wait for machine check
6. when machine check occurs hit PF1 to enter Mini Debug (if machine check does not occur, the stop address was not executed)

IBM Internal Use Only

To remove a previously set breakpoint the user must hit PF10 key while in memory display screen. This will cause the JIB instruction at the breakpoint address to be restored. After the breakpoint is removed JIB will resume normal execution.

When the breakpoint address is reached the process check light will start flashing.

To verify that the stop occurred where the breakpoint was set, the user should check PSWs in the register dump. One of the PSWs should be equal to the breakpoint set by the user.

IBM Internal Use Only

7.0 HAZEL WITH AP/CSS - DEBUG TOOL OVERVIEW

The Hazel Debug Tool (Debug) is available as part of the console AP's operating system. This tool provides the user with a comprehensive set of functions which will assist in development and debugging of both EDX and JIB code. Once the DSC has been IPLed, the user may access Debug at any time by hitting the PFI key. If the active task is sending information to the screen, the output operation will be completed before Debug is invoked. If, however, the active task is waiting for user input to the screen, the input operation will be enqueued, and Debug will be invoked immediately. Once Debug has been activated, it will assume control of the screen. An active task involving screen I/O will have its execution suspended until Debug has released control of the screen.

If two screens are attached to the DSC being used, the application being tested may use one screen, and Debug the other. In this case, an application program involving screen I/O will not have to compete with Debug for the use of a screen. Debug will be activated on the terminal on which the PFI key was hit.

Upon initial entry to Debug, the user is presented with a menu of primary commands (refer to Primary Commands Menu). Debug is driven by three types of commands: primary commands, secondary commands, and global commands. Primary commands initiate different types of activities. Examples of available activities include display and update of memory, display and update of registers, and stopping and tracing EDX or JIB code. Secondary commands are "second level" commands which perform functions that are specific to the activity selected by a primary command. These commands can not be accessed directly, but must be preceded by the appropriate primary command. Global commands set global parameters within Debug. These parameters define a format for screen output, as well as an area of HAZEL or AP memory to which memory oriented commands apply.

Most users will only be interested in a subset of the available functions. For users interested in operating system development involving JIB assembler code, Debug provides commands that will stop, single step assembler code, commands that will display and alter the contents of local storage and external registers, and a command that will display the contents of the system queues. For a description of the hardware relevant to development of this nature, refer to "DSC IV HARDWARE", by R.E. Pett. Users only interested in development of EDX code can ignore these functions entirely.

To become adept at using this tool, the user is encouraged to actually try the commands as they are covered in this manual.

IBM Internal Use Only

7.1 COMMAND ENTRY

All functions are initiated by user entered commands with or without associated parameters.

Commands are one or two characters in length, and parameters are either decimal or hexadecimal numbers up to four digits in length.

Whether a parameter is to be entered as a decimal or hexadecimal number depends upon the accompanying command.

Debug automatically pads zeros to the left of hexadecimal parameters less than four digits in length.

Both commands and parameters may be entered with any number of intervening blanks as long as no ambiguity results.

More than one command may be entered on the same line.

The following examples illustrate command and parameter entry and interpretation:

1.

PA 0000 0000 patches memory location x'0000' with the pattern: 0000.

P A 0000 0000 patches 3 implied sequential memory locations with the pattern: 000A 0000 0000 (refer to the "patch memory at current location" memory command).

PABCD 0000 patches memory location x'0BCD' with the pattern: 0000.

2.

M 9000 displays HAZEL memory at memory location x'9000'.

M9000 displays HAZEL memory at memory location x'9000'.

3.

SJ KS SM R? stops JIB, screen is not returned to EDX, sets display to scroll mode, and sets registers display mode.

8000 L22 PM displays HAZEL memory at memory location x'8000', sets display line count to 22, and sets display to picture mode.

IBM Internal Use Only

7.2 ARITHMETIC EXPRESSIONS

At any time, the user may use addition "+" and subtraction "-" sign between parameters to form arithmetic expressions. When Debug interprets the commands and parameters entered by the user, it generates a single equivalent parameter for each arithmetic expression it encounters. This may be useful when displaying or patching memory where offsets are involved. The following examples illustrate uses of special prompt characters:

OPERATION		EQUIVALENT
PA 8000+AA 0000	=>	PA 80AA 0000
P 1000+5 605-5	=>	P 1005 600
I -2	=>	I FFFE

Consider transferring 4AA words of memory (see "Transfer Memory" on page 200) from an AP starting address of 8FA0, to an AP destination address of 6000. Since the Memory Transfer command requires source start and end addresses, memory transfer could be performed by entering:

```
<cmd> <st_addr> <end_addr> <target>
T      8FA0      8FA0+4AA-1  6000
```

IBM Internal Use Only

7.3 GLOBAL COMMANDS

There are two types of global commands. One type is used to select an area of AP or HAZEL memory to which memory oriented commands apply. The other type is used to define a format for screen output. Global commands may be entered at any time; alone, or in conjunction with primary, secondary, or other global commands. The following is a list of global commands which define an area of memory:

CMD PARAMETER DESCRIPTION

HA	Switch to HAZEL mode. Focus all operations on the HAZEL CPU.
AP	Switch to AP mode. Focus all operations on the AP CPU.
CP <PARTN>	Change Partition. Address locations 0-7FFF are common to all three partitions. Valid partitions values range from 1 to 3 for Hazel/4 and 1 to 7 for Hazel/Ariel.
B= <ADDRESS>	Set HAZEL memory base offset. The specified base address is added to any HAZEL address specified when displaying HAZEL memory, or stopping EDX or JIB. Setting a base address is useful when debugging relocatable EDX programs.
DM <ADDRESS>	Display HAZEL memory on EDX stop. Unless the DM address is set to 0, on each EDX stop, stop information will include 8 words of HAZEL memory located at the specified address. Note, that the base address specified by the "B=" command is added to the address specified the "DM" command.
TS	Switch to SMALL EDX trace buffer size (256 words in the nucleus).
TL	Switch to LARGE EDX trace buffer size (24,576 words in the user memory A000 - FFFF partition 2). WARNING: LARGE trace will overwrite user memory A000 - FFFF in partition 2.

The following is a list of global commands which define a format for screen output:

CMD PARAMETER DESCRIPTION

PM	Picture Mode. Display new screen output beginning at the top of the screen.
SM	Scroll Mode. Display new screen output beneath the output which was last displayed.
R.	Do not display unchanged registers on a JIB stop. Periods "." are displayed in place of register pairs that have not changed since the previous JIB stop.
R?	Display all registers on JIB stop.
SK <#LINES>	Skip line. Redisplay the command prompt line a specified number of lines up or down. Valid values range from -7 to 7.
SP <#LINES>	Set memory display spacing. This command sets the number of blank lines to be inserted between blocks of memory displayed under scroll mode. Valid values range from 0 to 7. Entering "SP" alone, defaults to a spacing of 1.
L <LINE_CNT>	Set decimal number of lines of memory to be displayed. Valid line counts range from 1 to 99. The default of 4 lines can be set by entering a single "L". This command also sets the word count (refer to the "W" command) to a corresponding number of words.

IBM Internal Use Only

W <WORD_CNT> Set decimal number of words of memory to be displayed. When this command is used, subsequent scrolling will be by the specified number of words. For example, a WORD_CNT of 1 will cause the block of memory displayed to shift by one memory location each time the ENTER key is pressed. A WORD_CNT of 0 is useful for continually redisplaying the same block of memory. The word count is reset by entering a "L" command, or a "N" command without any associated parameters.

KS/FS Keep/Free Debug screen on EDX/JIB stop. The "KS" command keeps the Debug screen, and prevents the application screen from being restored between EDX/JIB stops. This command should only be used when the code between EDX/JIB stops does not require screen I/O. When the "FS" command is issued, the application screen is restored between EDX/JIB stops.

Note: The current global settings can be displayed by issuing the "G=" primary command.

IBM Internal Use Only

7.4 PRIMARY COMMANDS

7.4.1 Primary Commands Menu

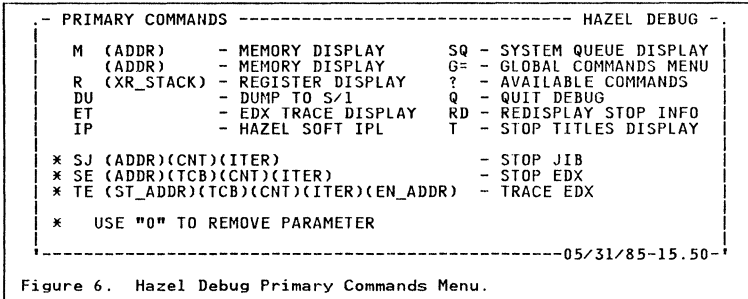


Figure 6. Hazel Debug Primary Commands Menu.

7.4.2 Display Current CPU's Memory

Syntax: "M <start_addr>"
Or: " <start_addr>"

This command causes the currently selected CPU's memory to be displayed in sequential lines containing the following information:

- the address with corresponding logical partition number if applicable,
- 8 double spaced words of memory in hexadecimal format, and
- a printable character representation of the 8 words of memory (unprintable characters are replaced with periods).

The "start_addr" parameter specifies the starting address from which memory will be displayed. Note, that if HAZEL is the currently selected CPU, the specified start address will be added to the base address set by the the "B=" global command. If "M" is entered alone, memory will be displayed beginning at the location last displayed.

7.4.3 Display the Currently Selected CPU's Registers

7.4.3.1 Hazel

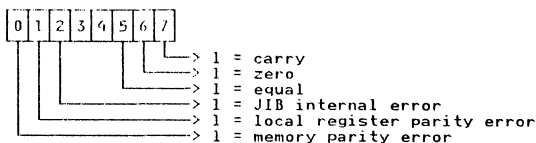
Syntax: "R"

This command causes the contents of the following registers to be displayed:

- 8 levels of local storage registers (LSRs) (16 registers per level).
- 32 external registers.
- 8 levels of program status words (PSWs). Three words per level are displayed in level order running from left to right, top to bottom:

IBM Internal Use Only

1. the instruction address register (IAR)
2. the condition codes and LSPRs
 - a. the first byte contains the JIB errors and condition codes:



- b. the second byte shows what LSPRs were assigned to a given level.



3. four low order nibbles of the following registers: #IREG, #DREG, #IRFG and #EPSW

HAZEL	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	RA	RB	RC	RD	RE	RF
LVL0 >	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
LVL1 >	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
LVL2 >	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
LVL3 >	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
LVL4 >	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
LVL5 >	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
LVL6 >	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
LVL7 >	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
SUPER >	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	X0	X1	X2	X3	X4	X5	X6	X7	X8	X9	XA	XB	XC	XD	XE	XF
	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0400 0001 2000				0727	0045	0001			3CF0	0001	0102				0902	0045 0103
37E0 0501 0204				3CF1	0045	0205			0E2B	0701	0322				3CEF	0501 0407

Figure 7. Hazel/4 Register Dump Screen.

7.4.3.2 AP/CSS

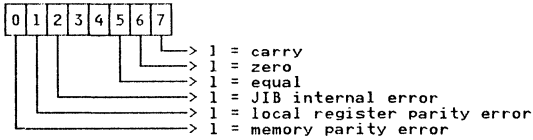
Syntax: "R <XR page>"

This command causes the contents of the following registers to be displayed:

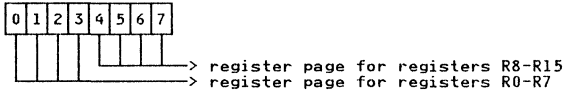
- 7 levels of local storage registers (LSRs)
 1. there are 16 registers per level
 2. the dump shows local storage registers on all levels even though, depending on the type of the AP/CSS, certain interrupt levels are not used
- 32 external registers for the currently selected page.
- 8 levels of program status words (PSWs). Two words per level are displayed in level order running from left to right, top to bottom:

IBM Internal Use Only

1. the first word in each pair is the current instruction address register (IAR)
2. the second word consists of two external registers:
 - a. the first contains the JIB errors and condition codes:



- b. the second shows what LSPRs were assigned to a given level.



The optional "XR_page" parameter is a number between 0 and 7 indicating which page of the AP's external registers are to be displayed. This parameter is ignored if Debug is in HAZEL mode.

AP	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	RA	RB	RC	RD	RE	RF
LVL0 >	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
LVL1 >	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
LVL2 >	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
LVL3 >	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
LVL4 >	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
LVL5 >	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
LVL6 >	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
PAGE: 1	X0	X1	X2	X3	X4	X5	X6	X7	X8	X9	XA	XB	XC	XD	XE	XF
	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0400 0001	0727 0045				3CF0 0001				0902 0045							
37E0 0501	3CF1 0045				0E2B 0701				3CEF 0501							

Figure 8. AP/CSS Register Dump Screen.

Figure 8. AP/CSS Register Dump Screen.

7.4.4 Dump Memory to S/1

Syntax: "DU"

This command is used to dump either AP or HAZEL memory to diskette. To initiate the DSC to S/1 dump, perform the following steps in sequence:

1. Select the desired CPU (HAZEL or AP) and partition using the appropriate global commands,
2. Enter the DU command,
3. Observe the AP light pattern - lights 0 to 3 should start flashing,
4. Determine the line number of the DSC which is required by the DTD program,
5. Insert the EDXDMP diskette into S/1 diskette drive, and

IBM Internal Use Only

6. Load the DTD program and enter requested information.

Hit **ENTER** if the dump abends, or to terminate the dump while it is in progress. The **ENTER** key disarms the dump and returns control to Debug. Refer to the "Appendix G. Memory Dump to S/I Diskette" on page 319 for further information.

7.4.5 Display System Queues

Syntax: "SQ"

This command displays information pertaining to the following system queues:

- Active Task
- Ready Queue
- Timer Queue
- CCB - ECB 1
- CCB - QCB 1
- CCB - ECB 2
- CCB - QCB 2
- Cancel Queue

The following information is displayed for each queue member:

1. the partition key,
2. the TCB address,
3. the contents of #1 and #2, and
4. the contents of 16 local storage registers.

If a queue is empty, the queue title will appear without any associated information. Display will halt after the display of each queue's information. To view the next queue's information, the user must hit the enter key. Queue information will eventually be redisplayed if the user continues to hit the enter key. The user may exit system queue display at any time by entering a new command.

7.4.6 Return to Previous Primary Subsystem

Syntax: "X"

This command is used to return the user to the previous primary subsystem. Up to eight previous subsystems may be accessed in succession by repeatedly specifying the "X" command. If the user attempts to back up more than eight levels, the primary commands menu will be displayed.

7.4.7 Display Global Settings

Syntax: "G="

This command is used to display the global commands menu and the current settings of the global parameters. The following is an example of the Global Commands Menu:

IBM Internal Use Only

GLOBAL COMMANDS		CURRENT SETTINGS
HA/AP	- HAZEL/AP MODE	HA
CP (KEY)	- CHANGE PARTITION (1 TO 3)	0001
B= (ADDR)	- SET BASE ADDRESS	0000
DM (ADDR)	- DISPLAY MEMORY ON EDX STOP	0000
TS/TL	- SHORT(256)/LONG(24,576) EDX TRACE	TS
PM/SM	- PICTURE/SCROLL MODE	PM
KS/FS	- KEEP/FREE DEBUG SCREEN ON STOP	FS
R?/R.	- DO/DO NOT DISPLAY UNCHANGED REGS ON STOP	R.
L (COUNT)	- SET LINE COUNT	0004
W (COUNT)	- SET WORD COUNT	0032
SP (LINES)	- SET DISPLAY SPACING (0 TO 7)	0000
SK (LINES)	- SKIP LINE(S) (-7 TO 7)	

Figure 9. Global Commands Menu and Current Settings.

7.4.8 Quit Debug

Syntax: "Q"

This command allows the user to quit Debug, and return to the application. If any JIB or EDX break points remain outstanding, Debug will continue to monitor execution, and will reactivate itself when a break point is reached. Similarly, JIB or EDX tracing will continue until the user re-enters Debug and explicitly requests that tracing be stopped.

7.4.9 Stop JIB

Syntax:

```
"SJ <stop_addr cnt iter |
    stop_addr cnt |
    stop_addr | >"
```

This command is used to stop HAZEL's execution of JIB code at a designated point. The following parameters specify where execution is to be halted:

stop_addr specifies the address at which execution is to be halted.

cnt specifies the maximum number of statements that will be executed following the statement at which HAZEL is currently stopped. When specified alone, this parameter defines the number of instruction that are to be executed before execution is halted. When specified in conjunction with a stop address, the count acts as a backup for situations where the execution path is uncertain. Valid values range from 1 to 256. This parameter is ignored if HAZEL is not stopped at the time of specification.

iter specifies the number of times the specified instruction is to be executed before HAZEL is stopped. Valid values range from 1 to 256.

Parameters may be omitted by placing zeros in the appropriate positions within the parameter list. If only zero(s) are specified, any outstanding break points are removed, the application screen is restored, and HAZEL is allowed to execute freely. If the "SJ" command is issued without any associated parameters, HAZEL is stopped on the JIB instruction currently being executed. A new break point may be specified at any time regardless of whether or not the previously specified break point has been reached. Since only one break point can be outstanding at a time, specification of a new break point will automatically remove the old break point. If HAZEL is stopped, the user may single step the code by hitting the enter key.

IBM Internal Use Only

This function is not capable of stopping on, or single stepping across a spi or a plex. Therefore, explicit break points must be placed before and after spis and plexes if the user wants to debug code involving such statements. Stopping on addresses which have interrupts disabled may only be done by first single stepping across the interrupt disable instruction. Similarly, before subsequently stopping on addresses which have interrupts enabled, the user must first single step across the interrupt enable instruction. Note, that the appropriate partition must be established using the "CP" global command when setting break points at addresses above 7FFF.

As each break point is reached, the following information is displayed:

- the partition key and address at which HAZEL is stopped,
- the instruction found at the stop address,
- 16 local storage registers,
- the condition code flags (E=equal, Z=zero, C=carry),
- the level, and
- the chapter and LSPRs.

A line of titles describing this information may be displayed by issuing the "T" command. If HAZEL is not stopped, this command is ignored. If the global command "R." has been issued, then only modified local storage register pairs are displayed following the first stop. Those which have not been modified, are shown to contain periods "." (refer to Global Commands). While HAZEL is stopped on an address, the user may execute any of the Debug commands. If the user exits "stop JIB" mode by executing a new primary command, the user must issue either the "X" (previous primary subsystem), or the "RD" (redisplay) command to reenter "stop JIB" mode. Upon reentering "stop JIB" mode, Debug will display the current "stop JIB" information, and then prompt the user for a "stop JIB" secondary command (refer to Stop Secondary Commands).

By default, the application screen will be restored upon the issuing of a "SJ" command. If no screen I/O is to take place before the break point is reached, the user may issue the "KS" (keep screen) command, causing the Debug screen to be kept (ie. as opposed to having the application screen restored). The user can again have the application screen restored between break points by issuing the "FS" (free screen) command.

7.4.10 Stop EDX

Syntax:

```
"SE      <stop_addr tcb_addr cnt iter |  
          stop_addr tcb_addr cnt |  
          stop_addr tcb_addr |  
          stop_addr | >"
```

This command is used to stop HAZEL's execution of EDX code at a designated point. The following parameters specify where execution is to be halted:

- stop_addr** specifies the address at which execution is to be halted.
- tcb_addr** specifies the task control block which is to be active when execution is halted on the specified address.
- cnt** specifies the maximum number of statements that will be executed following the statement at which HAZEL is currently stopped. When specified alone, this parameter defines the number of instruction that are to be executed before execution is halted. When specified in conjunction with a stop address, the count acts as a backup for situations where the execution path is uncertain. Valid values range from 1 to 256. This parameter is ignored if HAZEL is not stopped at the time of specification.

IBM Internal Use Only

iter specifies the number of times the specified instruction is to be executed before HAZEL is stopped. Valid values range from 1 to 256.

Parameters may be omitted by placing zeros in the appropriate positions within the parameter list. If only zero(s) are specified, any outstanding break points are removed, the application screen is restored, and HAZEL is allowed to execute freely. If the "SE" command is issued without any associated parameters, HAZEL is stopped on the EDX instruction currently being executed. A new break point may be specified at any time regardless of whether or not the previously specified break point has been reached. Since only one break point can be outstanding at a time, specification of a new break point will automatically remove the old break point. If HAZEL is stopped, the user may single step the code by hitting enter.

As each break point is reached, the following information is displayed:

- the partition key and address at which HAZEL is stopped,
- the contents of the index registers, and
- 8 words of memory specified by the "DM" global command.

A line of titles describing this information may be displayed by issuing the command "T". If HAZEL is not stopped, this command is ignored. While HAZEL is stopped on an address, the user may execute any of the Debug commands. While HAZEL is stopped on an address, the user may execute any of the Debug commands. If the user exits "stop EDX" mode by executing a new primary command, the user must issue either the "X" (previous primary subsystem), or the "RD" (redisplay) command to reenter "stop EDX" mode. Upon reentering "stop EDX" mode, Debug will display the current "stop EDX" information, and then prompt the user for a "stop EDX" secondary command (refer to Stop Secondary Commands).

By default, the application screen will be restored upon the issuing of a "SE" command. If no screen I/O is to take place before the break point is reached, the user may issue the "KS" (keep screen) command, causing the Debug screen to be kept (ie. as opposed to having the application screen restored). The user can again have the application screen restored between break points by issuing the "FS" (free screen) command.

7.4.11 Stop Titles

Syntax: "T"

This command is used to redisplay the stop information title associated with the current EDX or JIB stop (refer to the Stop EDX and Stop JIB commands). If the HAZEL is not stopped, this command will be ignored.

7.4.12 Redisplay Stop Information

Syntax: "RD"

This command is used to reenter "stop JIB" or "stop EDX" mode depending on which type of code is currently stopped. The stop information associated with the current EDX or JIB stop (refer to the Stop EDX and Stop JIB commands) is displayed upon specification of this command. If the HAZEL is not stopped, this command will be ignored.

7.4.13 HAZEL Soft IPL

Syntax: "IP"

AP/CSS first reinitializes its CSS channels and its internal control blocks. Then, it goes through the HAZEL IPL sequence. Depending on what

IBM Internal Use Only

is the IPL device HAZEL will be IPLed from EPROM card, AP/CSS or AP/DISK. The name of the nucleus loaded by AP/CSS is NC3 and by the AP/DISK is \$EDXIPL.

7.4.14 Trace EDX

Syntax: "TE <start_addr tcb_addr cnt iter stop_addr |
start_addr tcb_addr cnt iter |
start_addr tcb_addr cnt |
start_addr tcb_addr |
start_addr |>"

This command is used to trace a designated section of EDX code. The following parameters specify the section of code to be traced:

start_addr specifies the address at which tracing is to begin.

tcb_addr specifies the task control block which is to be active when tracing begins.

cnt This parameter defines the number of statements that will be traced following the the specified start address. If the start_addr parameter is not specified, this parameter defines the number of statements that will be traced following the statement at which the HAZEL is currently stopped. Valid values range from 1 to 256.

iter specifies the number of times the instruction at the specified start address is to be executed before tracing is to begin. Valid values range from 1 to 256.

stop_addr specifies the address at which tracing is to end.

Parameters may be omitted by placing zeros in the appropriate positions within the parameter list. If only zero(s) are specified, tracing is terminated, and HAZEL is allowed to execute freely. If no associated parameters are specified, EDX code is traced indefinitely beginning with the instruction currently being executed. New trace parameters may be specified at any time.

7.4.15 EDX Trace Size

Global commands TS (SMALL) and TL (LARGE) may be used to select EDX trace buffer size:

1. SMALL buffer - 256 words in the nucleus.
2. LARGE buffer - 24,576 words in the user memory (A000 to FFFF partition 2).

The default trace buffer size is SMALL.

7.4.16 View EDX Trace

Syntax: "ET"

This command is used to view the contents of the EDX trace table. The EDX trace is displayed in three or more columns: the first column contains a line number, the second column contains a TCB address, and subsequent columns contain "post", "chain", "TCB", and instruction address information. A post is indicated by:

- the letters "P__",
- the level from which the post was executed, and

IBM Internal Use Only

- the posted ECB address.

A chain is indicated by:

- the letters "C__",
- the level from which TCB was chained to the Ready Queue, and
- the associated TCB address.

The EDX trace will also display which instructions were executed by a given task. The TCB address is displayed in the second column, while associated instruction addresses are displayed sequentially from left to right in the next 7 columns. The user may scroll through the EDX trace as he would scroll through memory. Display of the EDX trace begins with the oldest part, whether or not it was generated by the most recent "TE" command. Scrolling in either direction will eventually wrap around and redisplay information that has already been viewed.

7.5 MEMORY SECONDARY COMMANDS

7.5.1 Memory Commands Menu

MEMORY COMMANDS

P (PATCH1)---(PATCH10)	- PATCH AT CURRENT LOCATION
P= (PATCH1)---(PATCH10)	- PATCH FOLLOWING LAST PATCH
PA (ADDR)(PATCH1)---(PATCH9)	- PATCH AT GIVEN ADDRESS
PR (PATCH)(COUNT)	- PATCH REPEAT
I (HEXCOUNT)	- INDEX UP/DOWN
S (WORD)(COUNT)	- SEARCH FOR WORD
T (FROM)(TO)(DEST)(DESTPARTN)	- TRANSFER CURRENT CPU'S MEMORY
	- SCROLL MEMORY
RS	- REVERSE SCROLL DIRECTION
X	- PREVIOUS PRIMARY SUBSYSTEM
=X	- PRIMARY COMMAND MENU
?	- AVAILABLE COMMANDS

Figure 10. Memory Commands Menu.

7.5.2 Patch Memory at Currently Displayed Location

Syntax:

"P <patch1 | patch1 patch2 | patch1 patch2 patch3 ...>"

This command is used to patch memory starting with the left, uppermost word in the block of memory last displayed. Up to 10 sequential words may be patched at a time. Intervening words which are not to be patched may be left untouched by inserting commas in the appropriate positions within the parameter list. For example:

P AAAA,BBBB,CCCC,DDDD will patch every other word beginning with the left, uppermost word in the block of memory last displayed.

Up to 10 words over a range of 16 words may be patched when using intervening commas.

7.5.3 Patch Memory at Following Last Patch

Syntax:

"P= <patch1 | patch1 patch2 | patch1 patch2 patch3 ...>"

This command is used to patch memory directly following the last patch made. Up to 10 sequential words may be patched at a time. Intervening words which are not to be patched may be left untouched by inserting commas in the appropriate positions within the parameter list (refer the "P" command).

Up to 10 words over a range of 16 words may be patched when using intervening commas.

7.5.4 Patch Memory at a Specified Address

Syntax: "PA address <patch1 | patch1 patch2 ...>"

IBM Internal Use Only

This command is used to patch the memory of the currently selected CPU starting at the address specified by the first parameter (plus a base address set by the "B=" command if patching HAZEL's memory). Up to 9 sequential words may be patched at a time. Intervening words which are not to be patched may be left untouched by inserting commas in the appropriate positions within the parameter list.

Up to 9 words over a range of 15 words may be patched when using intervening commas.

7.5.5 Patch Repeat Word in Memory

Syntax: "PR <pattern WORD_CNT | pattern | >"

This command causes the memory of the currently specified CPU to be changed according to the parameter(s) specified:

pattern specifies the word of data that is to be placed in memory (the default is x'0000').

WORD_CNT is a decimal number specifying how many words are to be replaced with the specified or default word (the default is a count of 1 word).

7.5.6 Index Through Memory

Syntax: "I <offset>"

This command allows the user to index through memory a specified number of words relative to the memory location currently displayed. The specified offset is assumed to be in hexadecimal. By prefixing the offset parameter with a minus sign "-", previous addresses may be referenced.

7.5.7 Search for Word in Memory

Syntax: "S <pattern WORD_CNT | pattern | >"

This command searches memory according to the the parameter(s) entered:

pattern specifies the word of data which is to be located (the default is x'0000').

WORD_CNT is a decimal number specifying the number of words, starting from the left uppermost corner of the block of memory last displayed, that are to be scanned for the specified pattern.

To continue searching following the last found pattern, reissue the "S" command without any associated parameters.

7.5.8 Transfer Memory

Syntax:

"T start_addr end_addr dest_addr <dest_partn>"

This command is used to copy a specified block of memory to another location either within one CPU or across CPUs. The source CPU, from which memory is copied, is defined by the current operating mode (either HAZEL or AP, global commands). If HAZEL is the source CPU then the current partition setting defines the source partition from which to copy. The following parameter descriptions explain how this command is used:

IBM Internal Use Only

start_addr	specifies the source start address of the block of memory to be copied.
end_addr	specifies the last address of the source block of memory.
dest_addr	specifies the starting address of the destination memory location.
dest_partn	if this parameter is specified, then HAZEL is the destination CPU.

7.5.9 Scroll through Memory

Syntax: " "

This command causes memory display to scroll in the direction currently specified.

7.5.10 Reverse Scrolling Direction

Syntax: "RS"

This command causes the display of the currently selected CPU's memory to reverse scrolling direction.

7.6 REGISTER SECONDARY COMMANDS

7.6.1 Register Commands Menu

REGISTER COMMANDS	
S (CHAPTER)(BLOCK)	- SELECT REGISTERS
T	- REGISTER TITLE DISPLAY
PR (LVL0-7)(REG0-F)(DATA)	- PATCH HAZEL LSRS
PX (XREG0-1F)(DATA)	- PATCH HAZEL XREGS
PX (XREG0-1F)(DATA)	- PATCH AP XREGS
X	- PREVIOUS PRIMARY SUBSYSTEM
=X	- PRIMARY COMMAND MENU
?	- AVAILABLE COMMANDS

Figure 11. Register Commands Menu.

7.6.2 Select Registers by Chapter and Block

Syntax: "S <chapter block | chapter | >"

This command is used to display HAZEL registers by chapter/block convention. The following parameters specify what information is to be displayed:

chapter specifies the chapter (0->7) from which registers are to be displayed (The default is chapters 0 through 3. If the user enters another "S" command, chapters 4 through 7 are displayed).

block specifies the pair of blocks within the specified chapter that are to be displayed (The default is all of the blocks: 0 through 7).

If an even numbered block is specified, that and the following block are displayed. If an odd numbered block is specified, that and the previous block are displayed.

7.6.3 Display Register Titles

Syntax: "T "

This command displays a line of titles for a chapter/block register display.

7.6.4 Patch Local Storage Registers

Syntax: "PR level lsr_no <patch>"

This command is used to patch local storage registers individually, or in pairs (even LSR followed by odd LSR). The following parameter descriptions explain how this command is used:

lvl_no lsr_no these two parameters specify which register is to be patched: the first parameter specifies the level (0->7), and the second parameter specifies the register (0->F).

patch specifies the patch to be made to the specified register.

IBM Internal Use Only

7.6.5 Patch HAZEL External Registers

Syntax: "PX xreg_no <patch>"

This command is used to patch HAZEL's external registers. The following parameter descriptions explain how this command is used:

xreg_no specifies which external register (0->1F) is to be patched.

patch specifies the patch to be made to the specified external register.

7.6.6 Patch AP External Registers

Syntax: "PX xreg_no <data>"

This command is used to patch AP's external registers one at a time. The following parameter descriptions explain how this command is used:

xreg_no specifies which external register (0->1F) is to be patched.

data specifies the patch to be made to the specified external register.

7.7 STOP SECONDARY COMMANDS

7.7.1 Stop Commands Menu

STOP COMMANDS	
* GE (ADDR)	- SINGLE STEP JIB/EDX
* PI (#1)(#2)	- GOTO EDX INSTRUCTION ADDRESS
= (ADDR)	- PATCH EDX INDEX REGISTERS
X	- DISPLAY MEMORY
=X	- PREVIOUS PRIMARY SUBSYSTEM
?	- PRIMARY COMMAND MENU
	- AVAILABLE COMMANDS
* - PERTAIN ONLY TO EDX STOPS	

Figure 12. Stop Commands Menu.

7.7.2 Single Step JIB/EDX

Syntax: " "

This command is used to single step JIB or EDX code depending on which type of code is currently stopped.

7.7.3 Go to User Specified Address

Syntax: "GE <addr>"

This command is used to change the HAZEL's instruction address register (IAR) to the specified address, and to halt execution when this address is reached.

7.7.4 Patch EDX Index Registers

Syntax: "PI <patch1 patch2 | patch1 | , patch2>"

This command is used to patch one or both of the EDX index registers. Note, that if only #2 is to be patched, the desired patch must be prefixed by a comma so as to leave #1 untouched.

7.7.5 Display Memory

Syntax: "= <addr>"

This command is used to display a specified or default area of memory. If an address is not specified, the memory last displayed will be redisplayed. Note, that global commands pertaining to memory display remain active. This command only displays an area of memory. To perform other memory functions, the user must enter "memory" mode by executing the "M" primary command.

IBM Internal Use Only

7.8 MISCELLANEOUS INFORMATION

7.8.1 Warning Messages

The following warning messages may appear on the Debug screen:

*** HAZEL IN M/C ***	This message indicates that Hazel process checked.
*** HAZEL INACTIVE ***	This message indicates that the VERIFY field in the APDVTB (see Figure 31 on page 259) is contaminated. In which AP/CSS will not perform any I/O work for Hazel because the APDVTB may be also contaminated.
*** DISK AP IN M/C ***	This message indicates that AP/DISK process checked.
*** DIS AP IN M/C ***	This message indicates that AP/DIS process checked.

7.8.2 Dates

Two types of dates in EBCDIC form may be found in the AP and Hazel nuclei:

- the assembly date (Hazel and AP)
- the last modification dates for all modules (AP only)

7.8.2.1 The Assembly Date

The date of assembly is the date when a given nucleus was assembled.

Hazel - use the supervisor utility function \$?.

AP/CSS - display the Debug's main menu (see "Primary Commands Menu" on page 190)

AP/DISK - use the DEBUG program to display AP/DISK memory at address X'0003'.

AP/DIS - use the DEBGA program to display AP/DIS memory at address X'0003'.

7.8.2.2 The Last Modification Dates

The date of last modification is given for each module:

AP/CSS - at address x'3F00'

AP/DISK - at address x'1380' (use the DEBUG program to display memory)

AP/DIS - at address x'1F00' (use the DEBGA program to display memory)

7.8.3 Machine Check

The following describes how AP/CSS a machine check conditions may be recognized in the following processors: Hazel, AP/CSS, AP/DIS and AP/DISK. In addition to this, it is described how a register dump can be obtained in order to determine the cause of a machine check.

IBM Internal Use Only

7.8.3.1 Hazel

If Hazel is in machine check a message: '*** HAZEL IN M/C ***' will appear, when the user enters Debug. To dump Hazel register the user must simply issue the register dump command - R.

7.8.3.2 AP/CSS

When AP/CSS machine checks it dumps its local and external registers on the 3101 and sounds a warning beep. Note, that the Debug cannot be used when AP/CSS is in machine check. The AP/CSS register dump is sufficient to determine the cause of a machine check.

```

      R0 R1 R2 R3 R4 R5 R6 R7 R8 R9 RA RB RC RD RE RF
LEVEL 0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
LEVEL 1 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
LEVEL 2 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
LEVEL 3 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
LEVEL 4 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
LEVEL 5 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
LEVEL 6 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AP PSWS : 00C0 0500 0200 0023 02B1 0145 048F 0567
AP PSWS : 048E 0000 0493 0089 284C 05AB 0CFF 0567
STACK 0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
STACK 0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
STACK 1 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
STACK 1 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
STACK 2 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
STACK 2 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

Figure 13. AP/CSS Machine Check Register Dump - Hazel/Ariel.

```

      R0 R1 R2 R3 R4 R5 R6 R7 R8 R9 RA RB RC RD RE RF
LEVEL 0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
LEVEL 1 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
LEVEL 3 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
LEVEL 4 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
LEVEL 5 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
LEVEL 6 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AP PSWS : 0A2A 0400 010E 0023 01B2 0000 01B3 0145
AP PSWS : 03A3 0067 03A7 0589 5000 05AB 0885 0401
STACK 0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
STACK 0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
STACK 1 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
STACK 1 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
STACK 2 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
STACK 2 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
STACK 3 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
STACK 3 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

Figure 14. AP/CSS Machine Check Register Dump - Hazel/4.

For the description of the AP/CSS register dump see "AP/CSS" on page 191. For a given stack, two rows of external registers are displayed in the register dump screen. The first row shows XR0 to XRF and the second row shows XR10 to XR1F.

IBM Internal Use Only

In the Ariel system, when in machine check, the AP/CSS turns on the Error Light LED which is located on the front panel of the card.

7.8.3.3 AP/DISK

To determine if AP/DISK is in machine check, the user must enter Debug. If AP/DISK is in machine check a message: '*** DISK AP IN M/C ***' will appear.

7.8.3.4 AP/DIS

When in machine check, the AP/DIS turns on the Error Light LED which is located on the front panel of the card.

If the front pannel of the card is not visible, the user may enter Debug to determine if AP/DIS is in machine check. If AP/DISK is in machine check a message: '*** DIS AP IN M/C ***' will appear.

When in machine check AP/DIS dumps its local and external register to Hazel's memory at location X'7F50':

X'7F50' - X'7F7F' local storage registers

X'7F80' - X'7F8F' PSWs

X'7FD0' - X'7FDF' external registers

The LEDs located on the top of the card show the #MCSTAT register. Since there are only four lights on the LED, four bits of the #MCSTAT register are displayed at time. The display sequence is as follows:

1. The LEDs flash the following pattern for four seconds:

1	0	0	1
---	---	---	---

 <—>

0	0	0	0
---	---	---	---

2. The LEDs are cleared.
3. Bits 0-3 of the #MCSTAT register are displayed for four seconds.
4. The LEDs flash the following pattern for four seconds:

0	1	1	0
---	---	---	---

 <—>

0	1	1	0
---	---	---	---

5. The LEDs are cleared.
6. Bits 4-7 of the #MCSTAT register are displayed for four seconds.

IBM Internal Use Only

8.0 TRACES

There are three types of traces available:

- EDX Trace
- Communications Trace - DSC
- Communications Trace - S/I

The EDX trace is described in detail in "Trace EDX" on page 197, "EDX Trace Size" on page 197, "View EDX Trace" on page 197 and in "Start/Stop EDX Trace - PF2" on page 181.

A detailed description of the communication protocol can be found in "Communication Transmission Subsystem" on page 275.

IBM Internal Use Only

8.1 COMMUNICATIONS TRACE - DSC

8.1.1 Communication Trace Buffer

The communication trace buffer is at location X'A000' in partition 3 or at location X'D000' in Hazel/3. The user may change the start of the trace buffer by patching the following:

- COMTPTR (see "Appendix B. Equate Tables" on page 293)
- value X'A000' in the following code in the module COMTRACE:

```
IF      (COMTPTR,GE,X'FFC0')
  MOVE  COMTPTR,X'A000'
ENDIF
```

The user may also change the partition by changing the value of the variable COMTRKEY in the module COMTRACE to a different value. The current communication trace pointer resides in the Nucleus Directory at location COMTPTR, see "Appendix B. Equate Tables" on page 293.

8.1.2 Enabling/Disabling the Communication Trace

To enable the trace the user must patch the GOTO statement in the module SUBSYS. See figure below:

TPLOOP	GOTO	TPLOOP2		TRACE SUPPORT (H4)
TPLOOP2	IF	(LCB+LCBMODE,EQ,+IDLE)	---	
	MOVE	LCB+LCBLSTAT,0		LINE INACTIVE
	WAIT	INT9		WAIT WITHOUT TIMEOUT
	ELSE		>	
	MOVE	LCB+LCBLSTAT,1		LINE ACTIVE
	WAIT	INT9,TIMEOUT=4000,P2=TIMER		WAIT WITH TIMEOUT
	ENDIF		---	

Figure 15. Module SUBSYS - Enable/Disable Communication Trace.

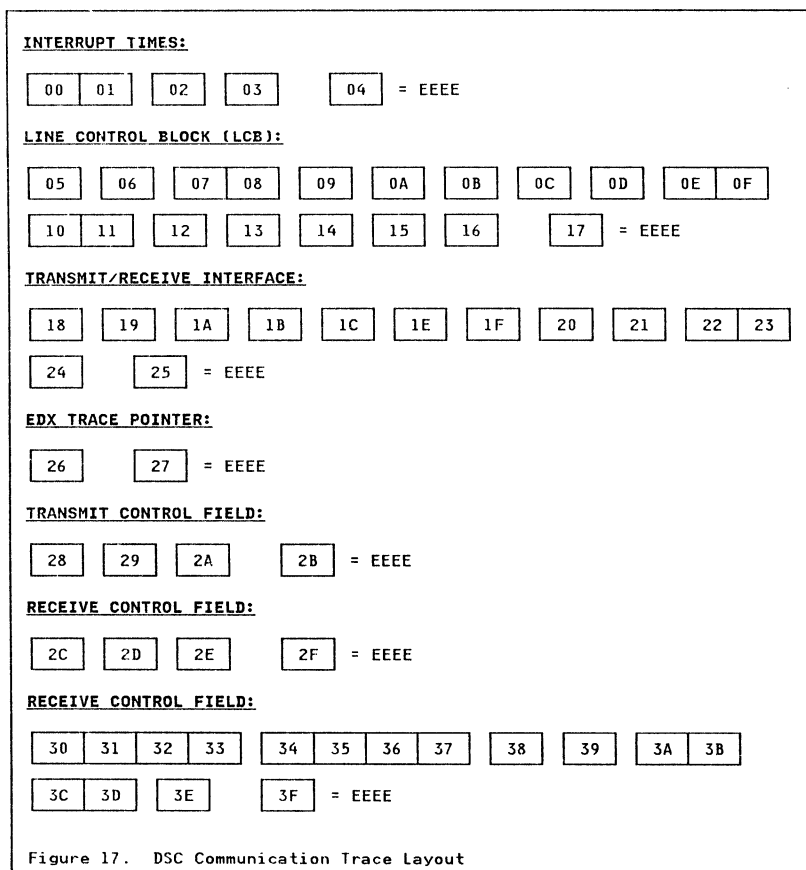
The GOTO address must be set to the address of the first EDX instruction in the COMTRACE module. See figure below:

AND	\$EDFLAGS,X'7FFF'	TURN OFF EDX TRACE	
MOVE	#1,COMTPTR		
INTIME	RELTIME,DELTIME		
CHKEY	COMTRKEY,KEYSAVE	CHANGE PARTITION	(H4)

Figure 16. Module COMTRACE - Enable/Disable Communication Trace.

To disable the trace the user must restore the GOTO address in the module SUBSYS.

8.1.3 Communication Trace Layout



8.1.4 Communication Trace Layout Description

- 00-01 - absolute time in milliseconds
- 02 - time since last interrupt in milliseconds
- 03 - time to service current interrupt in milliseconds
- 05 - line mode (high byte):
- 0 = idle
 - 1 = receive message
 - 2 = initial bid
 - 3 = sending/receiving message

IBM Internal Use Only

- 06 - our bid
- 07-08 - address of active send command (07 = address, 08 = key)
- 09 - message number
- 0A - acknowledge number
- 0B - error retry counter
- 0C - initial bid discard flag (high byte): 0 = do not discard, X = discard
- 0D - blast reset state (high byte):
 - 0 = normal operation
 - 1 = blast on next failure
 - 2 = last transmission was a blast reset
- 0E-0F - transmit delay end time in milliseconds
- 10-11 - address of send chain queue head (10 = address, 11 = key)
- 12 - next priority
- 13 - number of re-tries
- 14 - number of discards
- 15 - number of blast resets
- 16 - number of bad counts received
- 18 - our error on last transmit
- 19 - RS232 status byte:

0	1	2	3	4	5	6	7
---	---	---	---	---	---	---	---

- > overflow error - non-interruptible
 - > framing error - interruptible
 - > parity error - interruptible
 - > transmit buffer empty
 - > DSR
 - > receive buffer full
 - > interrupt pending
 - > CTS

}

on receive
- 1A - receive status
- 1B - LRC/CRC
- 1C - post code: FFFF - send command, FFFE - time out
- 1D - transmission type code (high byte):
 - 0 = not transmit
 - 1 = control field
 - 2 = control field and message
 - 3 = blast reset
- 1E - his bid for the line
- 1F - his time out (high byte):
 - 00 = he set a time out
 - 80 = he set no time out
 - 01 = unknown
- 20 - his message number
- 21 - his acknowledge number
- 22-23 - next transmit delay

IBM Internal Use Only

- 04B0 = (1200us) 1024 bytes at 9.6KB
 - 0007 = (7us) 6 bytes at 9.6KB
 - 0FA0 = (4sec) to wait for response
 - 0014 = (20us) to after normal short acknowledge end
- 24 - STIMER time
- 26 - EDX trace pointer
- 28 - control field 1: transmit message count
- 29 - control field 2:
- bit 0-3 = message number (0-F)
 - bit 4-7 = acknowledge number (0-F)
 - bit 8-F = his error
- 2A - control field 3:
- bit 0-7 = our bid
 - bit 0 = 0/1 our side has/has not a time out
 - bit 4-7 = acknowledge number (0-F)
 - bit 8-F = his error
- 2C - control field 1: receive message count
- 2D - control field 2:
- bit 0-3 = message number (0-F)
 - bit 4-7 = acknowledge number (0-F)
 - bit 8-F = our error
- 2E - control field 3:
- bit 0-7 = his bid
 - bit 0 = 0/1 other side has/has not a time out
 - bit 4-7 = acknowledge number (0-F)
 - bit 8-F = his error
- 30-33 - received message destination
- word 33: high byte = node/line, low byte = port number
- 34-37 - received message origin
- word 37: high byte = node/line, low byte = port number
- 38 - received message word count
- 39 - error on his last transmit
- 3A-3B - save Port Control Block (3A = address, 3B = key)
- 3C-3D - return buffer (3C = address, 3D = key)
- 3E - destination pointer

8.2 COMMUNICATIONS TRACE - S/1

The trace facility was developed to collect data for each send and receive on a selected IP line. Trace data can be stored in memory, to a diskette or to a hard disk. Four traces may be stored on a single diskette. Each data area on a diskette is 256 records in length, holding data for up to 512 transactions.

When a diskette or a hard disk option is used an overrun may occur. An overrun occurs when trace buffer is filled faster than it is emptied to a diskette or a hard disk. As a result, some trace transactions may be lost.

The advantage of using the the diskette option is that a S/1 failure will not delete collected data. The advantage of tracing to memory is that trace overrun will not occur. And finally, the advantage of using the hard disk option is that overruns are less likely than with the diskette option.

There are two versions of the trace: one with a 32-transaction buffer - TRACES and one with a 128-transaction buffer - TRACEB. The 128-transaction buffer trace requires more memory but overruns are less likely to occur.

See "Appendix H. S/1 COMMUNICATION TRACE" on page 327 for detailed description of the trace programs.

See "Communication Trace Layout" on page 210 for description of the trace content.

9.0 NUCLEUS GENERATION

The commands described in this section are used to define the particular configuration of a user's DSC. They are used only for creation of a nucleus and not during the creation of an application program.

*MODLISTING		
SYSTEM	MAXPROG=25	Start SYSTEM GENERATION
NUCTYPE	ENVIR=PROD	PROD / LAB
TERMINAL	3101,PAGSIZE=24,	Terminal definition
	LINSIZE=80,NHIST=12,	
	OVFLINE=YES	
\$SYSCOM	COMSIZE=2	System Communication area
COMM	ADDRESS=05,INTBIT=9	Host Communications
DISK	TYPE=IBM	DISK (optional)
HDWRTST	LRCHECK=NO	Hardware Bringup test
ENDSYS	COPY=NO	End SYSTEM GENERATION
COPY	NUCLEUS	
*MODLISTING END		
END		

Figure 18. Sample Nucleus Generation Statements

SYSTEM, ENDSYS and COPY NUCLEUS are required. The rest are optional and may be in any order but must be between SYSTEM and ENDSYS.

IBM Internal Use Only

9.1 NUCLEUS GENERATION COMMANDS

9.1.1 \$SYSCOM

Function

\$SYSCOM will generate a data area adjacent to the nucleus which may be accessed from any user application program. The size of this data area will be a multiple of 256 words. This common area is referenced indirectly in application programs through a storage location with the label \$SYSCOM. This label and the associated common area address reside in the program header and are generated by the PROGRAM statement. This storage location contains the address of the first word of the common area. Therefore, in order to reference data in the common area, the contents of \$SYSCOM should be loaded into a register, e.g. #1. Data elements may then be referenced by a displacement from this register.

The common area may contain Event Control Blocks (ECB), Queue Control Blocks (QCB), or any data blocks which must be accessed by more than one program. For example, if several programs must share the same data, this data is placed in the common area by an application program. Subsequent access to this data by other applications may be serialized by also placing a QCB in the common area and having the accessing program ENQ on this QCB before the access takes place.

Syntax

\$SYSCOM COMSIZE=

Required	COMSIZE=
Defaults	None

Operands

COMSIZE=

Specify the number of 256 word blocks to be reserved.

Notes:

1. SYSCOM resides in the nucleus partition and therefore its maximum size is controlled by the space left by the nucleus in the 32K word partition.
2. If no data communication area is desired, the user may either leave out the \$SYSCOM statement entirely or set COMSIZE=0.
3. The \$SYSCOM statement must precede the ENDSYS statement and follow the SYSTEM statement.
4. \$SYSCOM is not a command. The common area is automatically generated in the nucleus. It is the user's responsibility to allocate the use of this area and to ensure that access does not extend beyond this area.

The following example shows how to specify SYSCOM size.

```
$SYSCOM COMSIZE=2
```

The above \$SYSCOM statement will generate a 512 word (1024 byte) data communication area.

The following example shows how to use SYSCOM area. In order to reference a QCB located in the 15th word of the common area, the following instructions would be used:

IBM Internal Use Only

```
MOVE      #1,$SYSCOM
ENQ       (15,#1)
.         perform serial operation
.
DEQ       (15,#1)
```


IBM Internal Use Only

9.1.2 COMM

Function

COMM is used to specify the type and address of the device to be used for the EDX/J host communication support. Communications is supported by either the RS232 macrofunction card or by the AP_CSS attachment which emulates the RS232 operation. The COMM statement uses the TYPE= parameter to define the hardware so that the nucleus can be configured with the necessary support.

Only one COMM statement may be specified.

Syntax

```
label    COMM    TYPE=,ADDRESS=,INTBIT=
Required ADDRESS=,INTBIT= if TYPE=RS232
```

Operands

TYPE= Defines the communications hardware as either RS232 macrofunction, or CSS hardware. The statement should be coded as TYPE=RS232 or TYPE=CSS. If TYPE=RS232 is coded, then ADDRESS and INTBIT are required to be coded as well.

ADDRESS= A two hexadecimal digit LSA (Logical Space Address) defining the Block and Macrofunction address for the RS232 card being used.

INTBIT= A one digit hexadecimal value specifying the interrupt bit number (8-F) that is associated with the RS232C card.

Note:

If this statement is not present, HOSTCOMM and HOSTLOAD support code will not be generated in the nucleus. Example:

```
COMM ADDRESS=1A,INTBIT=3
```

IBM Internal Use Only

9.1.3 CPU

Function

CPU is used to specify whether the nucleus will be running on a 3 card engine or a 2 card engine (Hazel III or Hazel IV).

Syntax

label	CPU	CARDS=
Required		CARDS=
Defaults		CARDS=3

Operands

CARDS= Specify number of engine cards. (2 or 3)

Note: The CPU command is not required and a 2 card CPU is assumed.

IBM Internal Use Only

9.1.4 DISK

Function

The DISK statement specifies the need for Disk support in the nucleus to be generated. If no DISK statement is coded, no disk support is included. The disk support should only be specified for those hardware configurations which can support the disk: HAZEL/4 with a AP/DISK attachment.

Syntax

label	DISK	TYPE=
Required	TYPE=IBM	
Defaults	none	

Operands

TYPE= TYPE=IBM specifies the generation of a EDX/J nucleus which includes the IBM disk support.

IBM Internal Use Only

9.1.5 ENDSYS

Function

ENDSYS must be coded after the SYSTEM, HOSTCOMM, COMM, TERMINAL, etc. statements to designate the end of the nucleus generation commands.

This statement must be specified once and only once in the System Generation code. It must be followed either by an END statement or, if an application program is to be compiled with the nucleus, it may be followed by a PROGRAM statement.

Syntax

	ENDSYS
Required	None
Defaults	None

Operands

None

Note: If the post-processed listing is to include the JIB' microcode statements, this command should be coded as:

ENDSYS COPY=NO

and followed by the statement:

COPY NUCLEUS

IBM Internal Use Only

9.1.6 HDWRTST

Function

HDWRTST is used to specify whether the BRINGUP Test is to be performed. If this statement is missing, the BRINGUP code will not be generated.

Syntax

label	HDWRTST	LRCHECK=
Required		LRCHECK=
Defaults		LRCHECK=NO

Operands

LRCHECK= Specify whether the Longitudinal Redundancy Check (LRC) is to be performed. The LRC check is intended to verify whether or not the ROS to RAM load was successful.

IBM Internal Use Only

9.1.7 NUCTYPE

Function

NUCTYPE controls whether TCINT will be the first program automatically loaded and run. This statement requires one of the following keywords to define how the nucleus will operate after its IPLed: PROD, LAB, TEST.

Syntax

label	NUCTYPE	keyword
Require		PROD LAB TEST
Defaults		none

Operands

PROD

For production environment.

- On systems with 3270 keyboards and native displays: TCINT will be loaded automatically and run.
- On systems with 3101 terminals: TCINT will be loaded automatically and run. However, there is a feature that gives the operator the opportunity to bring the system up ready to be used with no predetermined load operation or prompt. This feature can be invoked for a 10 second period immediately after the nucleus is loaded by entering the supervisor utility function \$BP.

LAB

For laboratory environment.

- On systems with 3270 keyboards and native displays: If the operator presses the space bar during IPL he/she will be prompted for the name of the first program to be loaded and run. Otherwise, TCINT will be loaded.
- On systems with 3101 terminals: The operator will always be prompted for the name of the first program to be loaded and run.

TEST

For laboratory environment; the system is ready to be used and no predetermined load operation or prompt is desired.

IBM Internal Use Only

9.1.8 SYSTEM

Function

SYSTEM is used to define certain characteristics of the processor itself as well as system generation options. This statement must be specified once and only once, and should be the first statement of the System Generation code (with the exception of the COPY JIBPMAC statement which is used to invoke the microcode assembler).

Syntax

SYSTEM STORAGE=,MAXPROG=

Required	None
Defaults	STORAGE=64,MAXPROG=10

Operands

STORAGE=	Specify the storage size of the DSC processor module. The value must be a multiple of 16K, (i.e. 32, 48, or 64) where K=1024 words. A minimum of 32K is required and upgrading is done in multiples of 32K up to 128K.
MAXPROG=	Specify the maximum number of concurrently executing programs to be allowed by the system. Ten (10) is the default. The number of programs which can run concurrently in a system is a function of several variables, e.g. processor storage, program size, and processor time requirements, and, as such, will vary with each installation.

Examples:

SYSTEM	STORAGE=32,MAXPROG=10
SYSTEM	STORAGE=128,MAXPROG=5

IBM Internal Use Only

9.1.9 TERMINAL

Function

TERMINAL is used to specify which user Input/Output stations are attached to the system, along with their respective form parameters and special codes. Output only devices, such as line printers, are included under 'terminals'.

The DSC configuration and the EDX/J command set support either the RAM/Video macrofunction as the system output device and the native 3270 keyboard as the system input device or a 3101 terminal (controlled by the AP/CSS cards) as both the system input and output device. (This does not preclude the use of other input and output devices on the DIS channel. However, these devices would be completely controlled by the application.)

If an attempt is made to use the RAM/Video macrofunction when it does not exist in the configuration, a condition code of \$NOVIDEO will be placed in the task code word and the operation will act as a no-op.

It is assumed that the keyboard portion of the CONSOLE always exists, therefore input operations are performed whether or not the RAM/Video macrofunction exists on the DSC.

Syntax

label	TERMINAL	DEVICE=,RAM=,VIDEO=,PAGSIZE=,LINSIZE=, LINEDEL=,CHARDEL=,CRDELAY=,ECHO=,NHIST=,OVFLINE=
Required		RAM=,VIDEO=,(if RAM/VIDEO support is to be included; otherwise only 3270 support is included.)
Defaults		DEVICE=CONSOLE,PAGSIZE=16,LINSIZE=64,NHIST=12, ECHO=YES,OVFLINE=YES
label	TERMINAL	DEVICE=,PAGSIZE=,LINSIZE=,NHIST=,OVFLINE=
Required		none
Defaults (for Hazel 4 with AP/CSS)		DEVICE=3101,PAGSIZE=24,LINSIZE=80,NHIST=16, OVFLINE=YES

Operands

DEVICE=	Specify CONSOLE for the RAM/Video macrofunction on the DIS channel as the output device and the native 3270 keyboard as the input device.
RAM=	Specify 3101 for the 3101 terminal.
VIDEO=	Specify the two digit hexadecimal Logical Space Address of the RAM card to be used (if there is one). This operand should not be used if DEVICE=3101.
PAGSIZE=	Specify the two digit hexadecimal Logical Space Address of the Video card to be used (if there is one). This operand should not be used if DEVICE=3101.
LINSIZE=	This operand is used to define the page size (or 'form length') of the I/O medium. The value of this operand determines the effect of the SKIP and LINE options associated with output commands. Specify a decimal number between 1 and the maximum value which is meaningful for the device. For DEVICE=CONSOLE, specify 4, 8, 12, or 16 and up to 24 for DEVICE=3101.
	Specify the maximum length of an input or output line for the device. A value between 1 and 64 is used when DEVICE=CONSOLE and a value of up to 80 when DEVICE=3101.

IBM Internal Use Only

Note: The value of this operand must not exceed the physical limitations of the device being used. The results may be unpredictable.

NHIST=

For screen devices, this parameter specifies the number of history lines to be retained at the top of the screen when a page eject is performed. The lines retained are the last NHIST lines on the display prior to the page eject. The lines below the history area and down to PAGESIZE are the lines normally used by the terminal I/O instructions, thus providing a "rolling" effect to the terminal user. For DEVICE=CONSOLE, specify 0, 4, 8, or 12. NHIST can be in the range of 1 to PAGESIZE-1 when DEVICE=3101.

Note: NHIST must be less than PAGESIZE.

OVFLINE=

Code OVFLINE=YES if output lines which exceed the right margin are to be continued on the next line. If OVFLINE=NO is coded, output lines which exceed the right margin are truncated (extra characters are lost).

Display Mode

When using either a RAM/Video macrofunction or a 3101 terminal as the system logging device, the display will roll up automatically when data is to be placed on line PAGESIZE+1. The bottom NHIST lines on the display will move up to the top NHIST lines, and the new data is placed on the first line below the history area.

Example 1: Ram/Video display with 3270 keyboard

```
TERMINAL  DEVICE=CONSOLE, RAM=04, VIDEO=05
```

Example 2: 3101 Terminal

```
TERMINAL  DEVICE=3101, PAGESIZE=24, LINESIZE=80, NHIST=12
```

Note: If the terminal statement is not present, the support code will not be generated in the nucleus.

10.0 INTERNAL DESIGN GUIDE10.1 INTRODUCTION

This chapter contains information on the design and internal operation of the Event Driven Executive. It is intended for use by those who have a need to understand its operation.

10.2 OPERATING CONVENTIONS10.2.1 Common Setup Routine

Every EDX/J command is prepared for execution by a 'common setup' routine in the interpreter called \$CMSETUP. This routine performs a standard register setup and branches to the appropriate system subroutine required to execute the instruction. The system subroutine must return to \$CMSETUP with the registers R10 R11 containing the address of the next EDX/J instruction to be executed.

10.2.2 Register Conventions

The following register conventions are used in the common setup routine (\$CMSETUP):

Registers	Entry conditions	Exit conditions
-----	-----	-----
R4 R5	--	link register
R6 R7	Current TCB	same as input
R8 R9	--	Flags/Opcode
R10 R11	Current Command address	same as input
R12 R13	--	Address of operand 1
R14 R15	--	Address of operand 2

The contents of other hardware registers are dependent on the function being performed.

If indexing is used with either operand 1 or 2, the contents of the selected index register will be added to the register pair containing the operand address (R12R13 for operand1, R14R15 for operand2). If operand 3 is present, the subroutine '\$GETPAR3' may be used to obtain the address. Upon exit, R8R9 points to operand 3 (Result operand). R10R11 is incremented by 1 to account for the additional parameter. R0R1 are used by '\$GETPAR3' as work registers and linkage is via R4R5.

During assembly, a table of constants (average length of three words) is generated from each EDX/J command. The table has the following format:

Word	Contents
----	-----
1	Flag/Opcode
2	Address of Operand 1
3	Address of Operand 2
4	Address of Operand 3
5	Count

The first word of the table of constants generated during assembly is called the COMMAND WORD and has the following structure:

IBM Internal Use Only

Bit ---	Usage -----
0	Operand 2 type (1=constant,0=address)
1	Reserved
2-3	Register flag for operand 3
4-5	Register flag for operand 2
6-7	Register flag for operand 1
8-15	Operation code

The register flag for each operand may have the following values:

Flag ---	Meaning -----
0	Index Register not specified
1	#1 used in the form (X,#1)
2	#2 used in the form (X,#2)
3	#1 or #2 used explicitly

Example 1:

Assuming A is at address X'0100' and Z at X'0300', let us compile the following EDX instruction:

```

ADD (A,#1),1,RESULT=Z
      |   |   '----> OPERAND 3
      |   |-----> OPERAND 2
      |-----> OPERAND 1

```

After the compilation, the object code consists of four words:

```

8134 - Flag and Op Code
0100 - Address of A
0001 - Constant = 1
0300 - Address of Z

```

The Flags and Op Code fields have the following meaning:

```

X'8   ' - operand 2 is a constant.
X'1   ' - operand is index register 1 in the form (X,#1)
X'34' - operation code for single precision ADD with result

```

Example 2:

Assuming B is at address X'0100', let us compile the following EDX instruction:

```

ADD B,#2
      |   '----> OPERAND 2
      |-----> OPERAND 1

```

After the compilation, the object code consists of three words:

```

0332 - Flags and Op Code
0100 - Address of B
0001 - Index register 2

```

The Flags and Op Code fields have the following meaning:

```

X'03 ' - operand 2, register 2
X'32' - operation code for single precision ADD

```

When the flag bits = 3, the specific register (#1 or #2) is determined by the value of the appropriate operand in the parameter list. The operand values are 0 and 1 for #1 and #2, respectively (seen in Example 2).

There are 255 possible EDX/J instruction operation codes. An operation code of 0 implies a no-op. That is, an instruction whose command has bits 8-15 equal to zero, does not perform any operation. Registers R10 R11 are incremented by one to point to the next sequential word in memory. If the user wishes to no-op an instruction in an application program, zeros may be patched into each word of the instruction.

IBM Internal Use Only

10.2.3 Hardware Level Assignment of Hazel and APs

Level 0 is the highest priority level.

10.2.3.1 Hazel/3

Level Function

0	Machine check
1	Timers
2	3277 Keyboard
3	DIS interface
4	Not used
5	Application programs
6	Not used
7	Not used

10.2.3.2 Hazel/4 with Native Display

Level Function

0	Machine check
1	Timers
2	3277 Keyboard
3	DIS interface
4	Note used
5	Not used
6	Application programs
7	Not used

10.2.3.3 Hazel/4 with AP/CSS

Level Function

0	Machine check
1	Timers
2	Not used
3	DIS interface
4	Hazel-AP/CSS interface for 3101 and host communications
5	Not used
6	Application programs
7	Not used

10.2.3.4 Hazel/Ariel

Level Function

0	Machine check
1	Timers
2	Not used
3	Hazel-AP/DIS interface
4	Hazel-AP/CSS interface for 3101 and host communications
5	Not used
6	Application programs
7	Not used

10.2.3.5 AP/CSS

IBM Internal Use Only

Level Function

0	Machine check
1	AP/CSS-DEBUG program interface
2	Not used
3	Host communications interface
4	Secondary 3101 or ROC interface
5	Primary 3101 interface
6	Monitor
7	AP initialization

10.2.3.6 AP/DISK

Level Function

0	Machine check
1	AD/DISK-DEBUG program interface
2	Not used
3	Soft error and alarm reporting
4	Disk interface
5	Not used
6	Monitor
7	AP initialization

10.2.3.7 AP/CSS - ARIEL

Level Function

0	Machine check
1	AP/CSS-DEBUG program interface
2	Host communications interface
3	Primary 3101 interface
4	Not used
5	Not used
6	Monitor
7	AP initialization

10.2.3.8 AP/DIS - ARIEL

Level Function

0	Machine check
1	AP/DIS-DEBUG program interface
2	Not used
3	Local block interrupts
4	DIS I/O processor
5	Not used
6	Not used
7	AP initialization

IBM Internal Use Only

10.3 SUPERVISOR

10.3.1 General

The basic units in a multi-tasking system are tasks. A task is a group of instructions performing a predefined function. A program can consist of one or more tasks. The number of tasks in a program is dependant on the number of different operations that have to be relatively independent of one another.

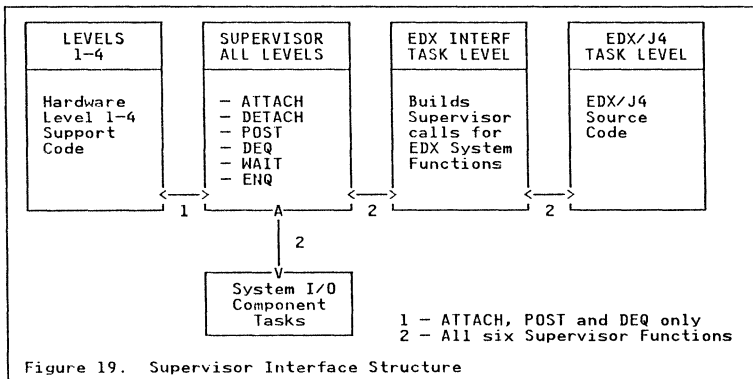
Associated with each task is a system generated table called the Task Control Block or TCB. The TCB contains a description of the parameters and current status of the task, along with save areas for the registers and work locations to allow the writing of reenterable routines.

Synchronization of tasks is performed by the supervisor which consists of six system functions and a task dispatcher running in a multi level environment. All tasks that are ready for execution are placed by the supervisor in a chain called the Ready Queue in a priority ordered manner (the highest priority task is at the front of Ready Queue). The task executing, known as the ACTIVE TASK, will be of a higher or equal priority than the task at the head of the Ready Queue. When the ACTIVE task stops executing for any one of a number of reasons (such as to wait for an event or a resource), it is removed from the Ready Queue by the TASK DISPATCHER and placed on a WAIT chain associated with the event or resource it is waiting for. The task at the front of the Ready Queue becomes the ACTIVE task and all tasks on this queue move up one position. In the quiescent state, the system will be running in the dispatcher wait loop, called the Idle Loop, until it is pulled out by an external interrupt.

The six system functions, mentioned above are listed here:

1. attach a task
2. detach a task
3. enqueue (reserve) a serial resource
4. dequeue (release) a serial resource
5. wait for an event
6. post an event

The supervisor functions run on four hardware levels and the task level (level 6). All six supervisor functions are supported at the task level. Only ATTACH and POST are supported at the hardware levels since it is not meaningful for an interrupt handler to issue a WAIT or an ENQ call, and by definition, only a task can DETACH itself (Figure 19).



IBM Internal Use Only

The supervisor functions are all reentrant, though in reality, reentry is only valid for ATTACH and POST. The supervisor itself is not reentrant. Reentry is provided through multiple supervisor work areas, one for each level.

Since the task dispatcher only runs on the task level, it can be called by the interpreter, the supervisor functions interface or by any systems components that run on that level. This simply dictates that the task switching control is limited solely to the task level (Figure 20).

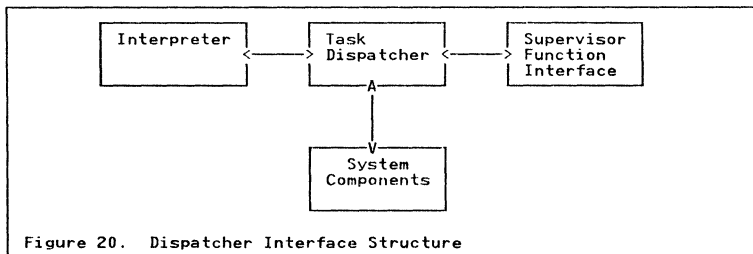
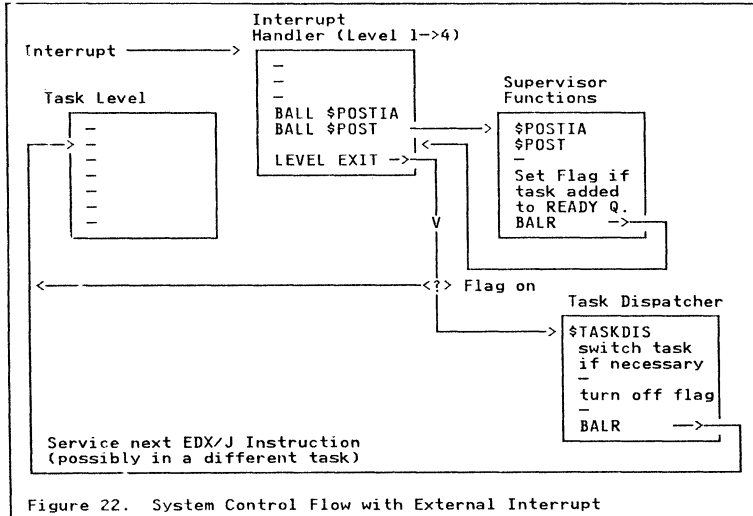
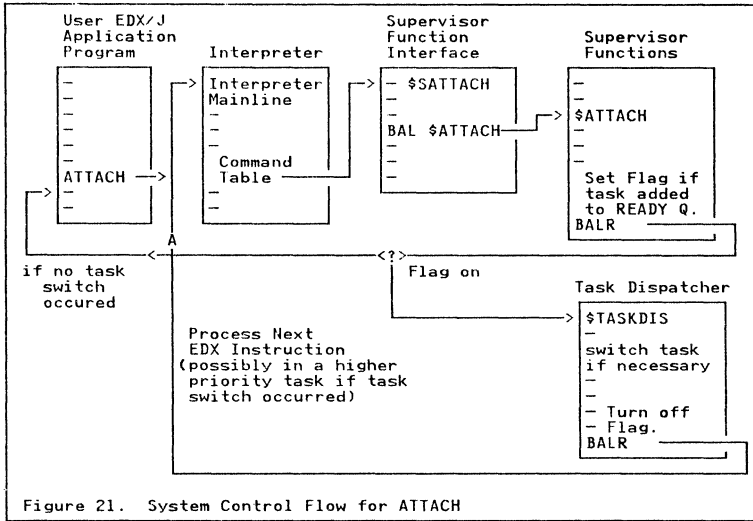


Figure 20. Dispatcher Interface Structure

Whenever an interrupt occurs there is the possibility of a task being moved to the Ready Queue which is of higher priority than the ACTIVE task. If this occurs then a task switch is required. Calls to the dispatcher at the hardware levels would provide optimal task switching, however, if the new tasks are generally of lower priorities, such calls would increase system overhead unnecessarily and degrade overall performance. The architecture allows the task dispatcher to be called only at the task level. This design reduces system overhead but not at the expense of task switching performance. External register 11 bit 0 is used as a flag which is turned on anytime tasks are moved to the Ready Queue. It is expected of all task level components - the interpreter, system components, supervisor function interfaces to examine this flag and call the task dispatcher accordingly. The task dispatcher will then compare the priorities between the ACTIVE task and the first task on the Ready Queue for possible task switching (Figure 21 on page 233 and Figure 22 on page 233).

IBM Internal Use Only



IBM Internal Use Only

10.3.2 Execution States

A task can be in one of four execution states:

- Detached** The task is in memory as part of the loaded program but is not running. This is the state all tasks are in when the program is first loaded. In this state the \$ATTACH function makes a task known to the system by placing it on the Ready Queue. Once attached, the task competes for system resources with all the other attached tasks. The contrary to the \$ATTACH is provided by the \$DETACH function. Once a task is attached, it is always in one of the three following states until it 'detaches' itself:
- Active** The task has been attached and is the currently executing on the task level.
- Ready** The task is on the Ready Queue indicating that it is ready to execute, but the CPU is under the control of a higher priority task or the system itself. Any number of tasks can be on the Ready Queue at the same time. When the CPU is available, the supervisor takes the task at the front of the Ready Queue (highest priority task) and sets it to the ACTIVE state.
- Waiting** The task is not ready to take over the CPU, because it is waiting for the occurrence of an event or the availability of a resource.

IBM Internal Use Only

10.3.3 Task Synchronization and Control

Communication between different tasks is made possible by the use of events and resources, known to all participating tasks. Each event is represented in the system by an ECB (Event Control Block), a two way switch which reflects whether the event has 'occurred' or has 'not occurred'. Each resource is represented by a QCB (Queue Control Block), which is also a two way switch reflecting whether the resource is 'available' or is 'busy'.

10.3.3.1 EVENT Control Block (ECB)

An ECB is the symbolic representation of a event. It is three words long.

The first word of an ECB, called the CODE FIELD, contains the status of the event. The event has not yet occurred if the code word is zero, while a non-zero in this field indicates it has.

The second and third words (address and key respectively) of an ECB make up the 'chain field'. The 'chain field' points to the first task on a list (chain) of tasks that are waiting on the event. By using a chain structure, the number of tasks (TCBs) which may be waiting for the same event is unlimited. The chain location (#TCBCHA) in each TCB on the chain points to the next TCB in the chain. The chain field of the last TCB in the chain contains a zero.

The difference between the Ready Queue and a WAIT chain is that there is only one Ready Queue in the system and it is at a fixed location. However there can exist many ECBs existing anywhere within memory.

A ECB chain with three chained elements (TCBs) is shown below.

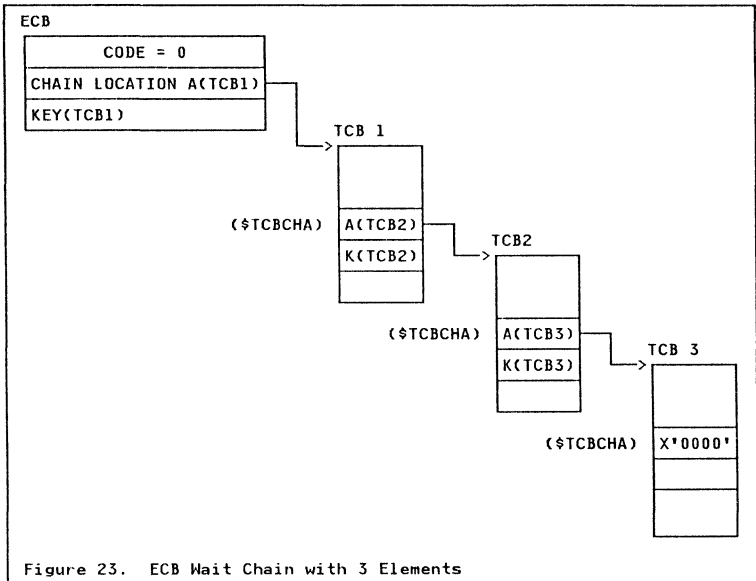
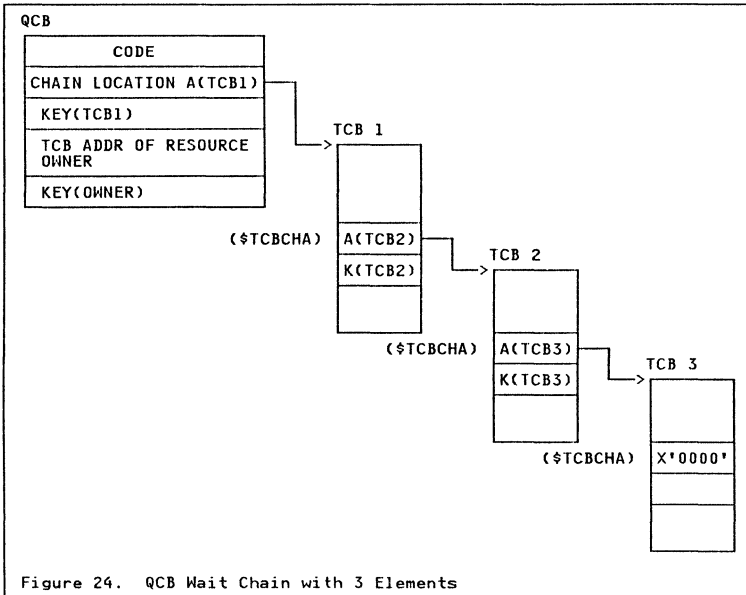


Figure 23. ECB Wait Chain with 3 Elements

10.3.3.2 Queue Control Block (QCB)

A QCB represents a resource. It shows the current resource status, its owner and the tasks waiting for it. It is five words long. The first word is the CODE FIELD, which contains a zero if the resource is busy. A non-zero value means it is available. The second and third words (address and key) make up the head of a chain of tasks waiting for the resource to become available. The chaining process is done in the same manner as described for the ECB. The fourth and fifth words (address and key) contain the address of the task that currently owns the resource.

A QCB chain with three chained elements (TCBs) is shown below.



IBM Internal Use Only

10.3.3.3 Task Control Block (TCB)

The TCB contains all the information necessary to describe the parameters and the current status of the task. The address of the main task control block is located in the Program Header (#PRGTCB). When there are multiple sub-tasks in an application program, the last task is pointed at by the main task (in #TCBPTR). Each task points at the previous one. Finally the first task points at the Main TCB address.

A TCB chain with 4 sub-tasks looks as follows:

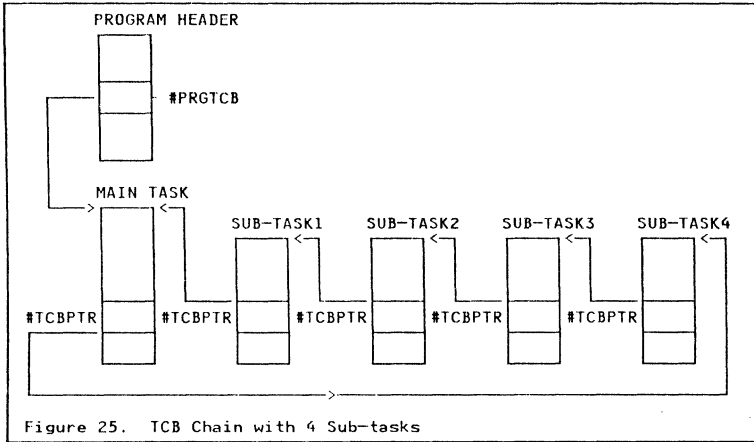


Figure 25. TCB Chain with 4 Sub-tasks

10.3.4 Functions

10.3.4.1 ATTACH

The ATTACH function takes a task that resides in memory but is not in any WAIT chain and places it on the Ready Queue. This command may be called either from a task or from the hardware level. Since a task must not be attached more than once (without a DETACH), the 'Task End' ECB (denoted EECB) of the TCB is tested. If the task is already attached, at the time an ATTACH command is issued, the EECB code word is equal to zero, indicating the task was not detached, and control is returned immediately to the calling routine resulting in a no-op.

During the ATTACHing process, four changes are made to the TCB of the new task. First, the EECB code field is cleared to indicate that the task is attached. The ATTACH code is placed in the TCB's code field. If no ATTACH code was present then -1 is used. The Console Control Block (CCB) address of the attaching task is inserted into the #TCBCCB field of the new task. Finally, the #TCBS4 field is loaded with the assembler link address.

The newly ATTACHED task is inserted into the priority ordered Ready Queue and waits there until it is dispatched and get controls of the system.

10.3.4.2 DETACH

The DETACH function removes the current task from the system, and therefore, cannot be called from the hardware level. Since a task can only detach itself, no TCB address is necessary in the parameter list.

The DETACH function POSTs the EECB of the task indicating that the task is no longer attached. If the DETACH code (any non-zero value) is not present, it is replaced by a '-1'. All TCBs which are waiting for this event are moved from the WAIT to the READY state. The Ready Queue changed flag is reset and the Task Dispatcher is called for a task switch.

If the task is attached again, control will be passed to the command right after the DETACH instruction.

10.3.4.3 WAIT

The WAIT and POST functions allow task synchronization by the use of events. If upon execution of the WAIT instruction (executable from the task level only), the event has not occurred (code field is zero), then the calling task is removed from the ACTIVE state and appended to the wait chain associated with the ECB. However, if the event has occurred by the time the WAIT is issued, then the WAIT acts as a no-op and execution proceeds sequentially.

10.3.4.4 POST

The POSTing of an event informs all tasks waiting for a particular event that it has occurred. The post code (default is -1) is placed in the code field of the ECB to indicate the event has occurred. All tasks that are waiting for the event are moved to the Ready Queue in a priority ordered fashion. If at the time of the POST, the ECB code field is already non-zero signifying the event has already occurred, then the status of the event is not changed and execution continues.

IBM Internal Use Only

10.3.4.5 ENQ

The ENQ and DEQ functions allow the use of not reentrant resources. A task that requires such a resource must be able to obtain and retain control until that resource is no longer needed. If it is not available at the time of request, then the requesting task must be prevented from proceeding until the resource becomes free. This is the purpose of the ENQ. The QCB associated with the resource is checked for availability indicated by a non-zero code field. If it is not busy then the requesting task becomes the owner. However, if it is busy, then one of two possibilities exists depending on the format of the ENQ instruction. If the BUSY= option of the ENQ command is used, then control is passed to the label specified in that field. If this option is not used, then the task is removed from the ACTIVE state and is placed on the wait chain of the QCB. In the case where the requesting task is already the owner of the resource, then no change takes place and control is passed to the command following the ENQ.

10.3.4.6 DEQ

Once a resource is no longer required it should be released so that it is available to other tasks. This is accomplished by a issuing a DEQ. If there are any tasks waiting for the resource when the DEQ is executed then the first task on the wait chain of the QCB becomes the resource owner and is moved to the Ready Queue. If on the other hand there is no one waiting then the resource is marked as 'available' (non-zero value in the code field) and control is given to the instruction right after the DEQ.

IBM Internal Use Only

10.3.5 TASK Dispatcher

10.3.5.1 General

The task dispatcher is a subroutine called on the task level to perform task dispatching.

A task switch is performed by saving the ACTIVE task's status and removing it from the ACTIVE state to either the WAIT state if it is waiting for an event or resource, or to the Ready Queue if a higher priority task is on the Ready Queue.

The dispatcher is called by the interpreter and System I/O components on the task level when the Ready Queue Modified Flag is set. The Ready Queue Modified Flag (External Register 11 Bit 0) is set whenever tasks are moved to the Ready Queue. It may therefore be set by the ATTACH, POST or DEQ functions. The Task Dispatcher may also be called by the WAIT, ENQ or DETACH functions when the executing task is removed from the ACTIVE state.

For calls from the interpreter or System Components, the calling task priority is compared with that of the first dispatchable task on the Ready Queue. If the calling task has higher priority, no task switching will take place, and control will return to the caller. Otherwise, the calling task is moved to the Ready Queue, and the first task in the Ready Queue would be dispatched. If the Ready Queue is empty, the dispatcher will be spinning in the Idle Loop until a task or tasks are moved to the queue.

10.3.5.2 Task Dispatcher Call

The task dispatcher is called as a subroutine and runs on the task level only. The dispatcher should be called whenever the Ready Queue Modified Flag is on. This flag is turned on whenever tasks are moved to the Ready Queue. To obtain optimal system throughput, the following components should examine the flag and if it is set, call the task dispatcher:

1. Interpreter:
 - If the bit is set, it calls the task dispatcher at the end of every EDX instruction.
2. Long EDX Instructions:
 - When the count exceeds 255, several EDX instructions (MOVE, MULT, FMULT, DIV, FDIV, SHIFTL, SHIFTR and IF), check the Ready Queue Modified Flag while executing. If the flag is set, they call the task dispatcher immediately.
3. I/O Support at the Task Level:
 - If the flag is set, it calls the task dispatcher after every ATTACH, POST or DEQ function.

Calling sequence:	BAL(L)	#TASKDIS
Input registers:	R4,R5	link register
	R6,R7	current TCB address
	R10,R11	current EDX command pointer
Output registers:	none	
Register integrity:	R0-R15	
Restriction:	task level LSPR points to the normal pages on entry	

IBM Internal Use Only

Note: In the event of a task switch, registers 0-15 would be saved in the ICB register save area associated with the caller. The task re-start address saved in R4R5 is the link used by the dispatcher. For any task that is dispatched, R0-R15 would be restored from the ICB register save area.

IBM Internal Use Only

10.3.6 Supervisor Service Routines

10.3.6.1 \$CHAINP

To insert a TCB into the priority ordered Ready Queue chain, the subroutine \$CHAINP is used. \$ATTACH uses \$CHAINP to chain new tasks while \$POST and \$DEQ use it to chain tasks from wait chains. \$CHAINP is used by the dispatcher during a task switch to chain the currently active task onto the Ready Queue.

10.3.6.2 REGSAVE

The subroutine REGSAVE is called by dispatcher on the task level and saves all the current task's registers in the register save area in its TCB.

10.3.6.3 SVSETUP

This subroutine is called by all six supervisor functions to set the register pages to those required by the supervisor and saves the caller registers in the supervisor control block.

10.3.6.4 LOADREG

LOADREG restores the caller's register that were saved away upon entry to the supervisor.

IBM Internal Use Only

10.3.7 Supervisor Work Area

The task supervisor uses a work area to keep information about the current state of the machine.

EDX/J Supervisor Work Area

\$TASKACT	DC	H'0'	Active Task
\$TASKKEY	DC	H'0'	Active Task Key
\$READYQ	DC	H'0'	Ready Queue Head
\$READYQK	DC	H'0'	Ready Queue Head Key
\$CODE	DC	H'0'	Code word
\$CURRTCB	DC	H'0'	Current Task
\$TEQCB			TCB/ECB/QCB Address
\$CURRKEY	DC	H'0'	Current Key
\$NEWKEY	DC	H'0'	New Key
\$SAVLSPR	DC	H'0'	LSPR Save Area
\$SAVLINK	DC	H'0'	Link Save Area
\$SAVIMSK	DC	H'0'	#IMSK Save Area

IBM Internal Use Only

10.3.8 Supervisor Function Calls

The six supervisor functions are called as subroutines by the use of branch and link instructions with the input parameters passed through registers. Since the functions are coded as subroutines, they are executed at the same hardware level as the calling routine.

The same routines handle both the hardware and task level functions. For those called on the task level, an interface routine is used to initialize the system for the execution of the function. This approach provides the same register integrity of EDX/J of all 16 registers for both hardware and task levels but does not require the duplication of support code. The LSPR must also be pointing to the normal pages when supervisor calls are made.

Shown below are the supervisor function call sequences, input parameters and register integrity for both the hardware and task levels.

10.3.8.1 ATTACH

Calling sequence: BAL(L) \$ATTACH

Input registers: R0,R1 ATTACH code
 R2 key
 R4,R5 link register
 R6,R7 TCB address
 R8,R9 address of TCB to be attached
 R10,R11 EDX/J command pointer

Output registers: none

Register integrity: R0-R15

ATTACH Code - The ATTACH code, inserted into the first word of the TCB of the task being attached, can be referred to by the task name. For a task that may be attached from more than one point, the code word can be used to indicate the origin of the attachment. Note that the code word should be examined immediately upon entry to the task since execution of certain commands (e.g. I/O) will cause it to be overlaid.

10.3.8.2 DETACH

Calling sequence: BAL(L) \$DETACH

Input registers: R0,R1 END ECB POST CODE (see EDX/J DETACH code)
 R2 key
 R4,R5 link register
 R6,R7 current TCB address

Output registers: none

Register integrity: R0-R15

DETACH Code - The DETACH code is used to post the END ECB of the detaching task. The END ECB indicates that a task has ended.

10.3.8.3 POST

IBM Internal Use Only

Calling sequence: BAL(L) \$POST

Input registers: R0,R1 POST code (see EDX/J POST code)
 R2 key
 R4,R5 link register
 R6,R7 TCB address
 R8,R9 address of ECB to be posted
 R10,R11 EDX/J command pointer

Output registers: none

Register integrity: R0-R15

POST Code - The POST code is inserted into the code field of the ECB representing a specific event. It must be a value other than zero since a zero indicates the event has not occurred. A non-zero code which informs waiting tasks that the event has occurred, may also be used as a flag to indicate a condition or a status.

10.3.8.4 WAIT

Calling sequence: BAL(L) \$WAIT

Input registers: R2 key
 R4,R5 link register
 R6,R7 TCB address
 R8,R9 ECB address
 R10,R11 EDX/J command pointer

Output registers: none

Register integrity: R0-R15

10.3.8.5 DEQ

Calling sequence: BAL(L) \$DEQ

Input registers: R0,R1 DEQ code (see EDX/J DEQ code)
 R2 key
 R4,R5 link register
 R6,R7 TCB address
 R8,R9 QCB address
 R10,R11 EDX command pointer

Output registers: none

Register integrity: R0-R15

DEQ Code - The DEQ code is inserted into the code field of the QCB which defines the resource. It must be a value other than zero since a zero indicates that the resource is not available. A non-zero code which implies a free resource, may also serve as a flag to indicate a condition or status.

10.3.8.6 ENQ

IBM Internal Use Only

Calling sequence: BAL(L) \$ENQ

Input registers: R2 key
 R4,R5 link register
 R6,R7 ICB address
 R8,R9 QCB address
 R10,R11 EDX/J command pointer

Output registers: none

Register integrity: R0-R15

IBM Internal Use Only

10.3.9 Immediate Action - (IA) for Hardware level

When the timing of external devices is not under program control, interrupts are used to synchronize these devices with programs. This is done within the framework of the supervisor functions previously described. Each hardware interrupt source is represented by an ECB in the system with the event being the interrupt itself. In order to pass the information that a device has generated an interrupt to the supervisor, there is entry point for each interrupt source which is executed each time the interrupt occurs. This routine is called an immediate action for hardware level and may be considered as the software extension of the hardware interrupt; in the simplest case, it only calls a POST function for the ECB representing the interrupt source. Therefore, a task may wait for the occurrence of a specific interrupt by calling a WAIT function for the ECB which is assigned to that interrupt.

10.3.10 Supervisor Call Routine - IA Level (Hardware level)

The only supervisor functions which may be executed in the immediate action are ATTACH, POST and DEQ. The registers of the caller are not saved by the supervisor. The entry points are \$ATTACH and \$POST.

10.4 INTERVAL TIMER SUPPORT

10.4.1.1 General

The interval timer support consists of two routines:

- EDX/J Command Service Routine (CSR)
- Interval Timer Interrupt Service Routine (ISR)

In general, the CSR queues the TCB of the requesting task onto the Interval Timer Queue. The ISR dequeues the TCB and posts the event when the time interval has expired.

The CSR services the following EDX commands:

- STIMER
- WAIT TIMEOUT=

with 3 entry points for:

- STIMER
- WAIT TIMEOUT=
- WAIT TIMEOUT= RESET

10.4.1.2 Interval Timer Queue

All tasks requesting a time interval are linked together on the Interval Timer Queue. The head of this chain is pointed to by \$TMICHAN,\$TMIKEY located in the Interval Timer Work Area. Other TCBs are linked through the TCB field referenced by the offset #TCBTCH. The TCBs are queued in time order; that is, the task requesting the shortest interval is first in the queue. This order is maintained by the CSR.

10.4.1.3 Time Interval

The time interval, in milliseconds, requested by a task is converted by the CSR to 50 usec units and stored in the #TCBTIM field of the TCB. Although this 4 byte field has the capacity for intervals up to 17 days, the present EDX/J commands limit the interval from 1 millisecond to 60 seconds in 1 millisecond increments. The CSR and ISR support the full 4 byte field.

10.4.1.4 Setting / Resetting the Interval Timer

The interval timer is set/reset by the CSR and ISR to time intervals from 50 usecs to 3 seconds in 50 usec increments.

ISR - Each time the timer interrupts, the elapsed time since the last reset is subtracted from the value of #TCBTIM of all TCBs in the Timer Queue. Those TCBs whose time interval has expired, are removed from the queue and if required, a event is posted. Otherwise, the elapsed time since last reset is placed in \$TMIVALU. The ISR then resets the interval timer with the smaller of:

- the 'time to go' for the first TCB in the Timer Queue
- 3 seconds

CSR - Figure 26 on page 249 describes some of the actions taken by the CSR when a task requests a timed interval. Figure 27 on page 249 describes how the CSR modifies the #TCBTIM field in the TCB and the exit path taken depending upon the EDX command.

IBM Internal Use Only

Timer Queue State	INACTIVE	ACTIVE	
#TCBTIM of requesting task	REQTIM	REQTIM + ET	
#TCBTCH	link	link	
\$TMIVALU	smaller of - 3 secs - REQTIM	REQTIM < TTG	REQTIM >= TTG
		REQTIM + ET	- no change
Interval Timer	started	restarted	- left running

ET - elapsed time - the time since timer last started
 TTG - time-to-go (before the running timer interrupts)
 REQTIM - time interval requested by task

Figure 26. Timer CSR Actions Part I

	STIMER	STIMER WAIT	WAIT TIMEOUT=
#TCBECB	0 0	0 A(TCB)	0 0
#TCBTOUT	0	0	A(ECB)
Exit	\$CMSETUP	\$TASKDIS	\$TASKDIS

Figure 27. Timer CSR Actions Part II

10.5 STORAGE MAP

The EDX/J nucleus uses storage maps to keep track of loaded programs and available storage. Each EDX/J partition is represented by its own storage map. Shown below is the structure of the storage map representing one of the 32K word partitions of EDX/J.

6000	first available location for loading
6006	PGM(1)
6614	PGM(2)
7A01	.
7B10	last program
FF00	unused space
FF00	unused space
FF00	unused space
FF00	unused space
FFFF	end of store map marker

Figure 28. Storage maps

The first word in the map contains the start address of the program load area.

The left byte of each entry in the storage map contains the high order address byte of the program load point (programs are loaded on 256 word boundaries so the low order byte of the program load point is always zero). The right byte of each entry contains the number of 256 word blocks of storage that are allocated to the program.

An unused entry is characterized by a '00' in the size byte while a 'FF' in this byte is the 'END of MAP' indicator.

The storage map is updated by both the \$GETMAIN and \$FREEMAIN routines. When an entry is to be removed and the map compressed, \$FREEMAIN is used. When an region of memory of given size is required, \$GETMAIN is used to search the storage map for an area large enough to meet the needs and to insert the appropriate entry.

The maximum number of program entries is specified by the MAXPROG parameter on the SYSTEM statement (see "SYSTEM" on page 224).

10.6 CONSOLE - TERMINAL I/O SUPPORT

The Console I/O support provides the interface between the Terminal I/O Instructions (READTEXT, PRININUM, etc) and the I/O support. The I/O support for the native display and 3270 keyboard system is the DIS Channel and for the 3101 system is the AP/CSS.

10.6.1 Organization

The Console I/O Support can be divided into 3 'layers':

1. Instruction, interpretation and attention handling
2. Code translation, numeric conversion and buffer management
3. Device support - this is the only layer of Console support which is device sensitive.

The Console I/O support code uses a control block known as the Console Control Block (CCB). The CCB which is similar in appearance and function to the TCB, defines a terminal instead of a task. Imbedded within the CCB are control words, counts, flags and pointers to describe the terminal to the system. Save areas also exist to allow the console support code to be reentrant. The console support uses the CCB to control the console be it two video macrofunction cards and 3270 keyboard or the 3101 terminal. Only one TERMINAL statement is allowed in EDX/J, therefore only one CCB can be built.

The console control block is generated by the TERMINAL statement when the nucleus is generated. EDX/J allows multiple TERMINAL statements, but the CCB is considered to be a serial resource. The console support must acquire and release control of this resource through the ENQT/DEQT facilities when servicing any console commands.

The console support describing Layers 2 & 3, refer to the native display and 3277 keyboard only. The AP/CSS Console Support used with the 3101 terminal uses Level 4 as an interface between the AP/CSS and HAZEL.

10.6.2 Layer 1

Layer 1 represents the interface between the user and either the EDX/J I/O support or the attention processor.

10.6.2.1 Instruction Interpretation

Entry Points:

```
* $SKLNSP - Line, Skip, and Spaces options
* $PRNTEXT - PRINTTEXT
* $PRNTNUM - PRINTNUM (single-precision)
* $PRNTNUM4 - PRINTNUM (double-precision)
* $GTVALUE - GETVALUE (single-precision)
* $GTVALU4 - GETVALUE (double-precision)
* $READTXT - READTEXT (word)
* $READTXL - READTEXT (line)
* $QUESTN - QUESTION
  $ENQT - ENQT
  $DEQT - DEQT
  $ENDATTN - ENDATTN
```

Listed above are the entry points of the EDX/J I/O commands. Each instruction marked with an asterisk will attempt to gain exclusive control of the CCB by calling the \$QUTERM routine. If the CCB required is available then execution of the command proceeds, otherwise the task is removed from the ACTIVE state and placed on the WAIT chain associated with the CCB.

10.6.2.2 Attention Handling

Entry Points:

```
$LEVEL2 - Start of Keyboard Interrupt Handler (systems with
$KEYWSH - Start of Attention Keyboard Task 3270 keyboard)
$ENDATTN - End of Keyboard Task
```

Depressing any key on the native 3270 keyboard causes an interrupt invoking the LEVEL2 (\$LEVEL2) interrupt handler. On the 3101, depressing a key invokes the AP/CSS Console Support code (to be discussed later). If the key depressed is the Attention Key, then the Attention Keyboard Task (\$KEYWSH) is attached by the interrupt handler. Once this task is attached the following actions are taken:

1. The Attention Keyboard Task attempts to gain control of the CCB by issuing an ENQT. If the attempt is successful, then execution proceeds, otherwise the task is placed on the WAIT queue associated with the CCB.
2. The operator, prompted by a '>' character, enters data.
3. The 'Systems Attention List' is searched to see if the entered data matches anyone of the System Attention List Commands.
 - If a match is found, then execution proceeds to the code supporting that particular command and exits by executing an ENDATTN command which releases its control of the CCB by a DEQT.
4. If the data does not represent a system function, it is determined if there is a program with its own Attention List and that list is searched for a match.
 - If there is a match, then execution proceeds to an associated entry point within the application program and exits by issuing a ENDATTN command.
 - If no match occurs, then the Attention Keyboard Task ends by executing the ENDATTN and a 'KEYWORD NOT FOUND' message is printed.

IBM Internal Use Only

10.6.3 LAYER 2

Layer 2 represents the buffer management and the data conversion support code which is transparent to the user.

10.6.3.1 Buffer Management

Entry Points:

\$PRTOUT - Output and Control	
\$CHPROC - Text Input	
\$VALSCAN - HEXADECIMAL and Decimal Input	EDX/J

The principal functions of \$VALSCAN, \$PRTOUT and \$CHPROC are:

1. Transfer of data between the user specified locations and the terminal I/O buffer (TBUF) in the CCB.
2. Invocation of the device support (LAYER 3) whenever transfer of data between the buffer and an I/O device is required (buffer full, new line indication, advance input exhausted, etc).
3. Recognition of the 'character delete' character and appropriate editing of the input line.
4. Scanning the buffer for, and removing, requested input (text, decimal or hexadecimal). Numeric data is delimited by calling \$VALSCAN.
5. Isolated values (hexadecimal or decimal) are converted to or from EBCDIC by conversion subroutines.

Return Codes:

Return Code	Description	Routine
X'00'	successful	all routines
-X'00'	bad return code	\$PRTOUT, \$CHPROC
X'FF'	value not found	\$VALSCAN
X'01'	omit value	\$VALSCAN
FLAG (R8 bit 7) = 0	more key entry	all routines
= 1	end of key entry	all routines
R9	Edit count	\$CHPROC

10.6.3.2 Numeric Conversion

Entry Points:

\$BDCWD - Binary to Decimal	(single/double precision)
\$HXBWD - HEXADECIMAL to Binary	" "
\$DCBWD - Decimal to Binary	" "
\$BHXWD - Binary to Hexadecimal	" "

Return Codes:

Return Code	Meaning	Routine
X'00'	successful	all routines
X'21'	double precision overflow	\$DCBWD
X'22'	single precision overflow	\$DCBWD
X'23'	overflow	\$HXBWD

The EBCDIC value to be converted is stored in #CCBFLD by the calling routine.

If an error occurs during EBCDIC to binary conversion, the binary target area is left unchanged, however if it occurs during binary to EBCDIC conversion, the EBCDIC field is generated to the point at which the error occurs.

For EBCDIC to binary, leading zeroes, although allowed, are ignored and processing begins with the first non-zero character ('+', '-' or a digit).

IBM Internal Use Only

For these reasons, the conversion routines would normally be used in conjunction with \$VALSCAN when converting variable format input data.

For binary to EBCDIC, hexadecimal fields are filled with zeros to the left and decimal fields are filled with blanks.

10.6.4 LAYER 3

10.6.4.1 Device Support Routines

Layer 3 contains the Device Support routines called from layer 2. This layer is used solely by EDX/J to handle the RAM/Video macrofunction and the native 3277 keyboard. Layer 3 is the only layer that is hardware dependent. There are three device Support Routines:

- \$L5KYWT - Level5 Keyboard Wait EDX/J
- \$SCRNMGR - Screen Manager EDX/J
- \$DPYPROC - Display Processor EDX/J

10.6.4.2 \$L5KYWT - Keyboard Wait Support

The function of this routine is to wait for one key to be entered from the native keyboard. When the key is depressed, the \$LEVEL2 Interrupt Handler posts the task waiting for the input data.

10.6.4.3 \$SCRNMGR - Screen Manager

Any data, either control or alphanumeric, that is to be written to the screen is processed by the screen manager. The screen manager has two functional parts which are:

1. Display function
2. Control function

The Display function, which is processed first, checks the caller's parameters in the CCB. If the parameters are acceptable, the screen manager computes the RAM address (on the video card) where the display data will start. All byte character counts and positions are converted to RAM word addresses and word data transfer counts. Once all the parameters have been set up, the Display Processor (\$DPYPROC) is called to perform the display operation.

The Control function handles screen formatting such as line changes and screen rolling along with the blanking the video RAM's storage. The Display Processor (\$DPYPROC) is used to send the control data to the screen.

10.6.4.4 \$DPYPROC - Display Processor

The Display processor formats the RAM/Video MDBs with the information contained in the CCB. All display operations, once started, continue until completely finished.

Each call to \$DPYPROC results in a 6-step operation to control the screen. Each of these steps requires a new MDB request to be formatted and processed by the DIS Channel support routines. In this section the RAM/Video MDBs are formatted from information contained in the CCB provided by the calling routine. Once a display operation is started, it will continue until finished and then return to the caller. The six control steps are:

1. Blank screen
2. Switch RAM from Share interface
3. Set RAM in-address incremented mode
4. Write display data
5. Switch RAM to Share interface
6. Unblank screen

The \$DPYPROC passes parameters to the DIS Channel Support by formatting an MDB. Four MDB fields are used:

IBM Internal Use Only

#MDBIDAT - IMM I/O data or I/O data pointer
#MDBLEN - data transfer length (words)
#MDBFLG - bit 0 Retry/no retry
 1 Read/Write
 2 DIS CHNL DISABLED
 4 IMM Read/Write
 7 Console operation
#MDBTECB - TCB address

See "MDB Description / Layout" on page 266 for a full description of MDB.

IBM Internal Use Only

10.7 AP/CSS CONSOLE SUPPORT

The AP/CSS Console Support provides an interface between Hazel and AP/CSS which is responsible for I/O on the 3101.

During the processing of a 3101 I/O command, a request is sent to the AP indicating that there is work to be done. This request is in the form of a Data Control Block (DCB). The DCB is built by either the \$PRTOUT or \$SKLNSP routines for output instructions (ie. PRINTNUM) or for the input instructions (ie. READTEXT).

Each I/O instruction may result in between one and four separate functions. For example, if the EDX command:

```
PRINTTEXT  '@MESSAGE'
```

is encountered, then three operations must occur to service the command ('@' is responsible for the first two):

1. if not empty, print contents of the screen buffer (TBUF)
2. move cursor to beginning of next line
3. print MESSAGE

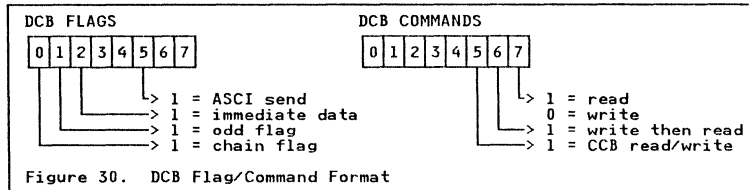
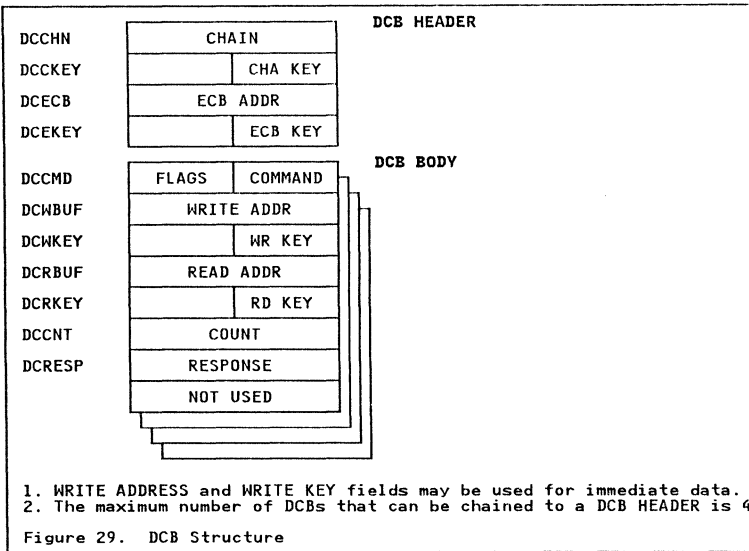
In this situation, a DCB is built for each function to be performed and all are concatenated to form a 'packet of work'.

At the front of each work packet is a four word DCB HEADER which has two functions:

1. It allows to chain other 'pockets of work' (CHAIN and CHAIN KEY fields). Note, that for 3101 support this function is not used, but it may be used for other I/O devices in the future.
2. It identifies the owner of the DCB by it's ECB and ECB KEY fields. Note, that for 3101 support the ECB field is the CCB address. The CCB contains the ECB to be posted after each packet of work has been completed.

Shown below is a DCB HEADER header with four DCBs.

IBM Internal Use Only



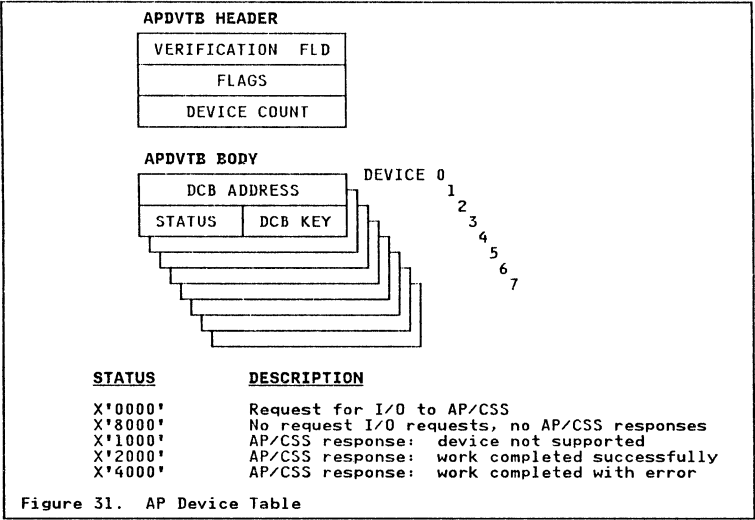
After a 'packet of work' has been constructed, it is chained to the appropriate entry (the 3101 device) in the AP DEVICE TABLE (APDVTB see Figure 31 on page 259). The 3101 device 'pockets of work' are always chained to the APDVTB itself, in other devices chaining to already chained 'pockets of work' may be allowed. The APDVTB can accommodate up to eight devices. Each device entry in the table is two words long:

- The first word is a pointer to a DCB header.
- The second word contains the status (see Figure 31 on page 259) of the DCB pointed to by the first word and the key of the DCB header.

After the 'packet of work' is chained to the APDVTB, AP/CSS is interrupted and it starts searching the APDVTB status fields for the 'new work' code (X'0000'). Meanwhile, the task that issued the I/O request is placed in the WAIT state. After CSS/AP executes a 'packet of work', it interrupts Hazel on level 4 and sets the status field in the APDVTB to one of the three possible values of the return code. Hazel level 4 scans the status fields in the APDVTB searching for valid return codes. When a device with a valid return code is found the ECB specified in the DCB header chained to that device is posted with the return code specified in the APDVTB status field. Finally, the status field is set to X'8000'; this is to state that this

IBM Internal Use Only

device has neither any work for AP/CSS nor response to be processed by Hazel.



IBM Internal Use Only

10.7.1 Level 4 Interrupt Handler

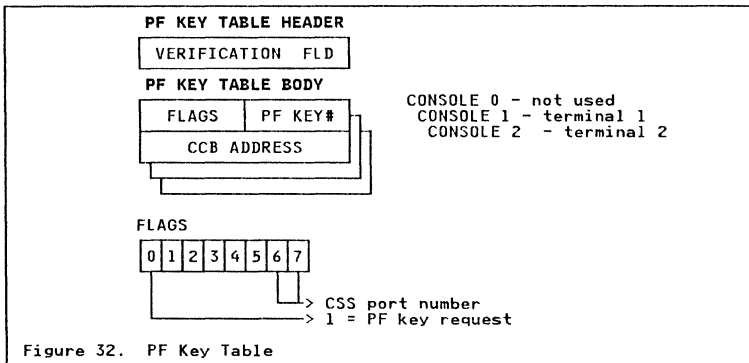
Level 4 interrupt handler is activated by the AP/CSS to perform one of the following services:

1. AP/CSS Debug request
2. I/O Interrupt Table request
3. PF key request
4. 3101 I/O completed response

Item 1 and 2 is not related to AP/CSS Console Support and item 4 was discussed in the previous section. Item 3 - the PF key request will be discussed here.

The PF key requests from AP/CSS are identified by checking the PF key request flag in each of the three PF Key Table (see Figure 32) entries. If the flag indicates a presence of a PF key request then following operations are performed:

1. The CCB's ECB is posted with the post code set to the PF key number.
2. The PF key request flag is reset.



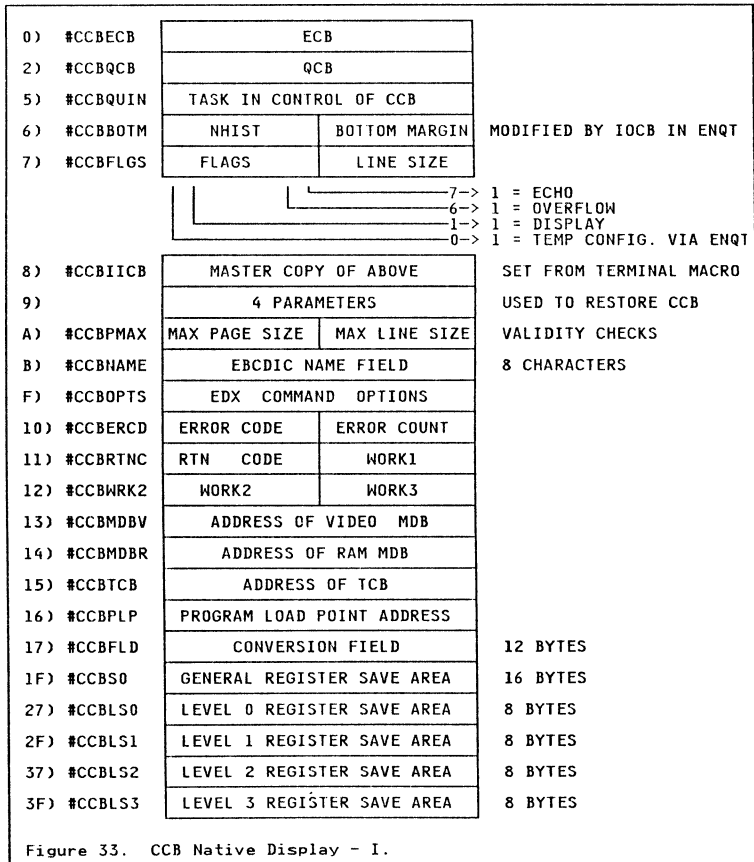
IBM Internal Use Only

10.7.2 Console Control Block (CCB)

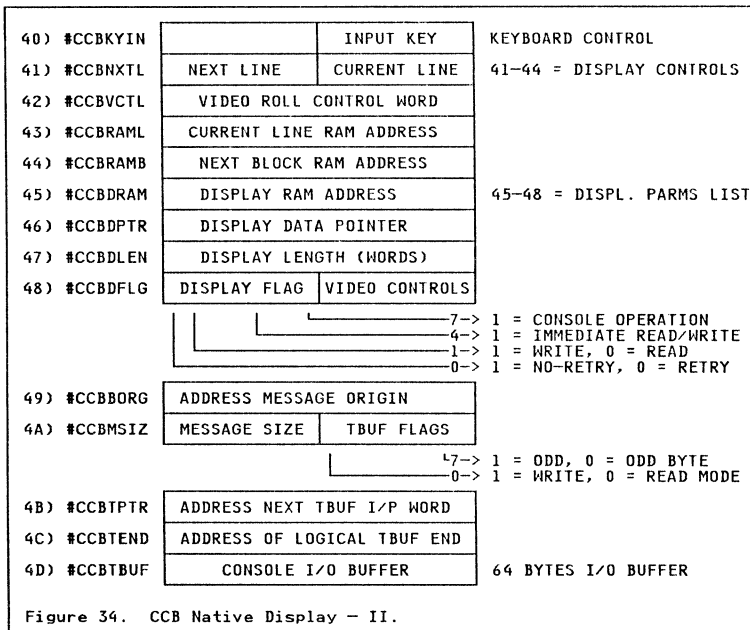
The TERMINAL statement causes a Console Control Block (CCB) to be included in the supervisor at \$SYSLOG. The CCB serves the following purposes:

1. Define particular device (console) characteristics
2. Provides console component addressing
3. Contains links to other system control blocks or modules (address of the task in control of the terminal)
4. Provides Buffers (data storage and conversion areas) as well as control fields for device support (FMS video control and video RAM cards).

Note: The console control block for the native display is quite different (as one might expect) from the console control block for the 3101. The layouts of both are shown below.



IBM Internal Use Only



IBM Internal Use Only

0)	#CCBECB	ECB			
3)	#CCBQCB	QCB			
8)	#CCBQUIN	TASK IN CONTROL OF CCB			
B)	#CCNIICB	MASTER COPY OF ABOVE 4 PARMS			SET FROM TERMINAL MACRO
D)	#CCBPMAX	MAX PAGE SIZE	MAX LINE SIZE		VALIDITY CHECKS
E)	#CCBNAME	EBCDIC NAME FIELD			8 CHARACTERS
12)	#CCBOPIS	EDX COMMAND OPTIONS			
13)	#CCBERCD	ERROR CODE	ERROR COUNT		
14)	#CCBRTNC	RTN CODE	WORK1		
15)	#CCBWRK2	WORK2	WORK3		
16)	#CCBMDBV	SPARE			4 BYTES
18)	#CCBFLD	CONVERSION FIELD			12 BYTES
1E)	#CCBS0	GENERAL REGISTER SAVE AREA			16 BYTES
26)	#CCBLS0	LEVEL 0 REGISTER SAVE AREA			8 BYTES
2B)	#CCBLS1	LEVEL 1 REGISTER SAVE AREA			8 BYTES
30)	#CCBLS2	3101 DEVICE OFFSET			
31)	#CCBNXTL	NEXT LINE	CURRENT LINE		DISPLAY CONTROLS
32)	#CCBVCTL	SPARE			3 BYTES
35)	#CCBD RAM	DISPLAY RAM ADDRESS			35-38 = DISPL. PARMS LIST
36)	#CCBDPTR	DISPLAY DATA POINTER			
37)	#CCBDLEN	DISPLAY LENGTH (WORDS)			
38)	#CCBDFLG	DISPLAY FLAG	VIDEO CONTROLS		
39)	#CCBBORG	ADDRESS OF MESSAGE ORIGIN			39-3C = I/O CONTROLS
3A)	#CCBMSIZ	MESSAGE SIZE	TBUF FLAGS		
3B)	#CCBTPTTR	ADDRESS NEXT TBUF I/P WORD			
3C)	#CCBTEND	ADDRESS OF LOGICAL TBUF END			
3D)	#CCBTBUF	CONSOLE I/O BUFFER			80 BYTES
8D)		CHAIN HEAD			8D-90 = DCB CONTROLS
8E)			CHAIN KEY		
8F)		ECB ADDRESS			
90)			ECB KEY		

Figure 35. CCB 3101 - I.

IBM Internal Use Only

91)	FLAGS	COMMAND	91-97 = COMMAND BLOCK 1
92)	WRITE ADDRESS		
93)		WRITE KEY	
94)	READ ADDRESS		
95)		READ KEY	
96)	COUNT		
97)	RESPONSE		
98)	FLAGS	COMMAND	98-9E = COMMAND BLOCK 2
99)	WRITE ADDRESS		
9A)		WRITE KEY	
9B)	READ ADDRESS		
9C)		READ KEY	
9D)	COUNT		
9E)	RESPONSE		
9F)	FLAGS	COMMAND	9F-A5 = COMMAND BLOCK 3
A0)	WRITE ADDRESS		
A1)		WRITE KEY	
A2)	READ ADDRESS		
A3)		READ KEY	
A4)	COUNT		
A5)	RESPONSE		
A6)	FLAGS	COMMAND	A6-AC = COMMAND BLOCK 4
A7)	WRITE ADDRESS		
A8)		WRITE KEY	
A9)	READ ADDRESS		
AA)		READ KEY	
AB)	COUNT		
AC)	RESPONSE		

Figure 36. CCB 3101 - II.

Figure 36. CCB 3101 - II.

IBM Internal Use Only

10.8 DISTRIBUTED INTERFACE SYSTEM (DIS)

10.8.1 General

EDX/J application programs can access macrofunction cards on the DIS Interface by the DISWRITE and DISREAD commands. Programs using these commands must first define the macrofunction card(s) to the system with the LSA or the MDCB statement.

The LSA statement provides input for generating the Macrofunction Data Block (MDB) which is built by the MDB macro when invoked by the ENDPROG statement. In programs assembled for Hazel/Ariel, LSA or MDCB statements may be used. The LSA statements generate MDCBs which are built by the MDB macro when invoked by the ENDPROG statement. The LSA statement does not allow chaining of MDCBs. The MDCB statement causes the MDCB macro to generate the MDCB structure inline. The MDCBs generated by the MDCB statement can be chained.

The INIBIT statement generates Interrupt Data Blocks (IDBs). Its function is to define interrupts from the DIS that are to be serviced by the application program. The IDB is built by the IDB macro when invoked by ENDPROG. The IDB is really an Event Control Block (ECB) with an extra field containing a 4 character EBCDIC name derived from the label coded for the source statement.

Macrofunction cards and interrupting sources on the DIS interface are considered to be serial resources, therefore, each one can have only one MDB, MDCB or IDB linked to it by the operating system. During a program load, each IDB in the program is checked to see if it has been previously defined to the system. If there are no IDBs already defined, then the load ties them into the IDB Pointer Table. However, if a repetition does exist, then the loader aborts. Note, that the loader does not check if the MDBs or MDCBs have been previously defined to the system.

IBM Internal Use Only

10.8.2 MDB Description / layout

If the DISWRITE or DISREAD command is coded without declaring a 'loc' parameter address, the default command flags are formatted to indicate that the immediate data input/output area is to be used. The immediate input/output data area may be referenced in the application program in the same manner as any other variable. This allows the user to use the MDB structure for the I/O buffer.

0) #MDBIDAT	Immediate Data/ Data Address		
1) #MDBUV	U - BITS	V - BITS	
2) #MDBCTC	RCTC	CTC	NOTE 1
3) #MDBLEN	I/O Transfer Length (Words)		
4) #MDBLSA	LSA Address	Flags	NOTE 2

47->	1 = CONSOLE OPERATION
4->	1 = IMM READ/WRITE
1->	1 = WRITE, 0 = READ
0->	1 = RETRY

5) #MDBTECB	TCB.ECB Address		
6) #MDBCHN	MDB Chain Word		EDX/J
7) #MDBRTNC	Return Code		NOTE 3
8) #MDBSAV1	General Save Area		
9) #MDBMUV	MASTER U BIT	MASTER V BIT	NOTE 4
A) #MDBTYPE	Macrofunction Type Code		

#MDBSIZE - MDB Table Entry Length

Figure 37. MDB Format

NOTE 1: The read back CTC (RCTC) is only valid for a DISWRIT command when U/V bits are coded. However, this field is also used to indicate when the U/V BITS are to be used in the execution of the DISWRIT and DISREAD. The rule is: if RCTC = 0 the U/V BITS are ignored, otherwise, they are used. The DISWRIT and DISREAD macro assembler supports the above rule.

NOTE 2: The immediate READ/WRITE DATA FLAG is set when no I/O BUFFER is defined and the caller expects the MDB #MDBIDAT field to be used in the transfer. If flag BIT 4 is set and the I/O count (#MDBLEN) is greater than 1, the bytes in #MDBIDAT are repeatedly written out until the count is depleted or only the last two bytes are retained for a read operation.

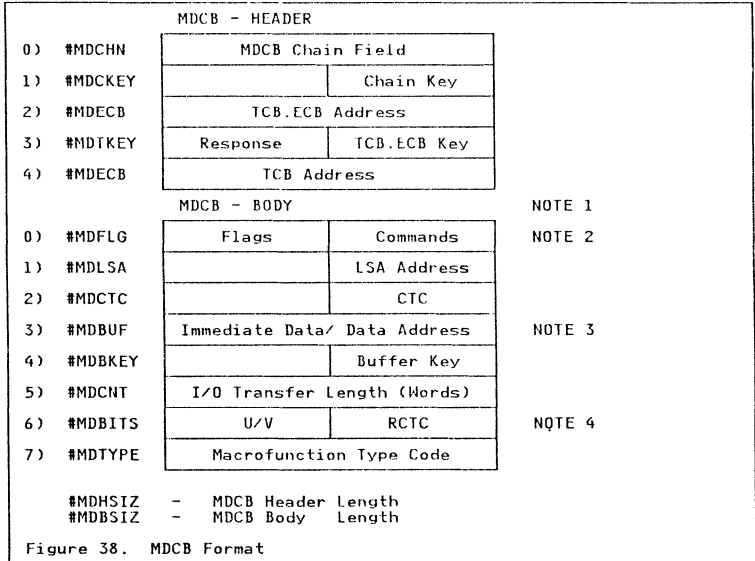
NOTE 3: #MDBTECB is used to save the TCB address.

NOTE 4: #MDBMUUV is set from the U/V BITS entered for the LSA macro. These bits are defined in the MDB and are used as the default controls if the DISREAD or DISWRITE command does not provide the U/V bits information. The defaults for the MASTER U/V FORMAT BITS are for full 8 bit (odd CTC) or 16 bit (even CTC) transfers.

IBM Internal Use Only

10.8.3 MDCB Description / Layout

If the DISWRITE or DISREAD command is coded without declaring a 'loc' parameter address, the default command flags are formatted to indicate that the immediate data input/output area is to be used. The immediate input/output data area may be referenced in the application program in the same manner as any other variable. This allows the user to use the MDB structure for the I/O buffer.



NOTE 1: In the figure above only one MDCB body is shown. In reality, the number of MDCB bodies that can be attached to a single MDCB header is unlimited.

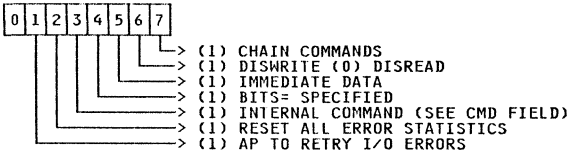
NOTE 2: See Figure 39 on page 268.

NOTE 3: The IMMEDIATE DATA FLAG is set when no I/O BUFFER is defined and the caller expects the MDCB #MDBUF field to be used in the transfer. The MDCB #MDBUF field should never be referenced via the #MDBUF equate, because of the effects of the MDCB header and MDCB chaining. The immediate data should always be accessed via the label defined by the P2= parameter on the DISREAD, DISWRITE and MDCB commands. If flag BIT 5 is set and the I/O count (#MDCNT) is greater than 1, the bytes in #MDBUF are repeatedly written out until the count is depleted or only the last two bytes are retained for a read operation.

NOTE 4: The read back CTC (RCTC) is only valid for a DISWRIT command when U/V bits are coded. However, this field is also used to indicate when the U/V BITS are to be used in the execution of the DISWRIT and DISREAD. The rule is: if RCTC = 0 the U/V BITS are ignored, otherwise, they are used. The DISWRIT, DISREAD, and MDCB macro assembler supports the above rule.

IBM Internal Use Only

FLAGS



COMMANDS

0	1	2	3	4	5	6	7		
0	0	0	0	0	0	0	0	→	NO OPERATIONS PERFORMED (NOOP)
0	0	0	0	0	0	0	1	→	START POLLER (POLLON)
0	0	0	0	0	0	1	0	→	STOP POLLER (POLLOFF)
0	0	0	0	1	0	0	0	→	READ POLL LIST AND STATISTICS (READP)
0	0	0	0	1	0	0	0	→	UPDATE POLL LIST (UPDATEP)
0	0	0	1	0	0	0	0	→	READ ERROR LOG (READE)
0	0	1	0	0	0	0	0	→	CARD LIST (CARDL)
0	1	0	0	0	0	0	0	→	READ ERROR LOG (FCARD)
1	0	0	0	0	0	0	0	→	EXECUTE RAM CODE AT X 2000 (EXECRAM)

Figure 39. MDCB Flags and Commands

IBM Internal Use Only

10.8.4 IDB Description / Layout

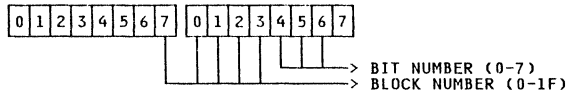
The IDB contains an ECB to be used by those operations which require the supervisor to service an interrupt and 'POSI' an operation complete. These Support Macrofunctions are capable of interrupting: analog input (AI), process interrupt (PI), and digital I/O with external sync (DI/DO).

0)	#IDBECB	ECB - code field		
1)		ECB - chain field		
2)		ECB - key	ECB validation	
3)	#IDBADR	Interrupt Bit Address		NOTE 1
4)	#IDBNAM	Interrupt Bit Name		NOTE 2
5)	#IDBGECB	General ECB Address		
6)	#IDBGKEY		ECB Address Key	

#IDBSIZE - IDB Table Length

Figure 40. IDB Format

NOTE 1: The Interrupt Bit Address is encoded as follows:



NOTE 2: Only the numeric part of the name is stored in this field.

10.9 IPL BRING UP TESTS / INITIALIZATION**10.9.1 Bring Up Tests - Hazel**

The BRINGUP tests verify all JIB instructions, all memory, local and external registers, level switching, parity and nucleus LRC before the operation begins.

The general approach of checking the instructions is to verify each instruction before it is used to verify any others except for the BRANCH instruction. Since there is no other way to hang the machine other than a branch to itself, it is necessary to assume that the BRANCH instruction works properly.

At the beginning of each test, an error condition indicator is displayed on the DATA BUS OUT (DBO) lights. If for some reason the test is not completed then the lights will remain on showing the code (shown below) of the test in which the failure occurred. If all the tests are completed successfully then the lights are turned off.

LIGHTS	Hazel/3
x'00'	Register-Immediate instruction
x'01'	Register-Register instruction
x'02'	External Register-Immediate instruction
x'03'	SRL or LDR instruction
x'04'	Nucleus LRC failed to match
x'05'	Control Store Direct instruction
x'06'	Control Store Indirect instruction
x'07'	BALR or JUMP instruction
x'08'	Local Register
x'09'	Interrupt Level Switching
x'0A'	High Memory Initialization to X'FF'
x'0B'	Memory tests

Figure 41. DBO Light Settings for Hazel/3 Bringup Errors

LIGHTS	Hazel/4 or Hazel/Ariel
x'00'	Hazel/4 instruction tests failed
x'01'	Nucleus LRC failed to match
x'1x'	Control store instruction tests failed
x'2x'	Local register tests failed
x'30'	Nucleus is not a Hazel IV nucleus
x'4x'	Interrupt level switching tests failed
x'5x'	Extended memory addressing tests failed
x'6x'	Memory tests failed
x'7x'	Memory tests failed
x'8x'	Timer interrupt test failed
x'9x'	External register tests failed

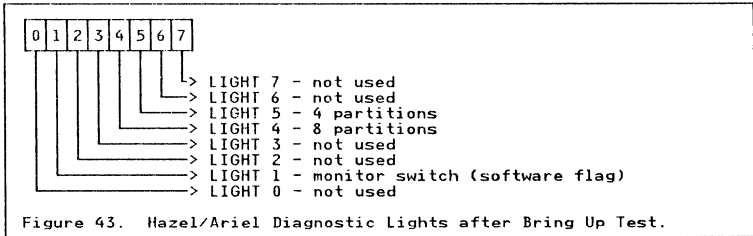
Figure 42. DBO (Hazel/4) or LED (Hazel/Ariel) Light Settings for Bringup Errors

Hazel/Ariel does not have DBO lights. The diagnostic lights are displayed on LEDs which are mounted on the Hazel/Ariel engine card. To see the LEDs one must remove the top cover of the machine.

The run error LED on the engine card is lit during the Bring Up Test. When the Bringup Up Test is completed, the run error light is turned off.

IBM Internal Use Only

After the Bring Up Tests are completed some of LED lights are used by the operating system. See figure below:



IBM Internal Use Only

10.9.2 Initialization

After the BRINGUP test has been completed, the INITIALIZATION module is invoked. Its function is to initialize the hardware and various software data structures of the nucleus. Shown below is the initialization procedures for Hazel.

10.9.2.1 Hazel/3, Hazel/4 and Hazel/4 with AP.

1. Initialization of all local storage registers and PSWs
2. Screen initialization:
 - a. macrofunction card set up
 - b. blank display
 - c. DIS channel initialization
 - d. CCB initialization
 - e. get AP/CSS Device Table address (not in Hazel/3)
3. Initialization of Arithmetic Processing Unit (APU)
4. Clear out TRACE TABLE
5. Clear AP Communication Area (7F00)
6. Insert Machine Check Handler Address (\$LEVEL0) into first 3 words of the nucleus
7. Calculate the maximum EDX partition value and store it in \$MAXPART (not in Hazel/3)
8. SPI to Interpreter Level
9. Interpreter level initialization:
 - a. Set flags in APDVTB control block to signal APs that Hazel has completed the initialization. (not in Hazel/3)
 - b. ATTACH communications line task (\$LINE01)
 - c. ATTACH initial program load task (\$LNLOAD)
 - d. ATTACH initial program load task (\$S1TASK) (not in Hazel/3)
 - e. ATTACH any program assembled with nucleus
 - f. go to Idle Loop

10.9.2.2 Hazel/Ariel

1. Initialization of all local storage registers and PSWs
2. Screen initialization:
 - a. CCB initialization
 - b. get AP/CSS Device Table addr
3. Initialization of Arithmetic Processing Unit (APU)
4. Clear out TRACE TABLE
5. Clear AP Communication Area (7F00)
6. Insert Machine Check Handler Address (\$LEVEL0) into first 3 words of the nucleus
7. Calculate the maximum EDX partition value and store it in \$MAXPART
8. SPI to Interpreter Level
9. Interpreter level initialization:
 - a. Turn off the Run Error Light to signal APs that Hazel has completed the initialization.
 - b. ATTACH communications line task (\$LINE01)
 - c. ATTACH initial program load task (\$LNLOAD)
 - d. ATTACH initial program load task (\$S1TASK)
 - e. ATTACH any program assembled with nucleus
 - f. go to Idle Loop

IBM Internal Use Only

10.9.3 Bring Up Tests - AP's

The BRINGUP tests checkout all JIB instructions, all memory, local and external registers, level switching and cycle-steal interface.

The general approach of checking the instructions is to verify each instruction before it is used to verify any others except for the BRANCH instruction. Since there is no other way to hang the machine other than a branch to itself, it is necessary to assume that the BRANCH instruction works properly.

At the beginning of each test, the Run Error Light is turned on. If for some reason the test is not completed then the light will remain on and LED's mounted on the AP card will show the code (shown below) of the test in which the failure occurred. If all the tests are completed successfully then the Run Error Light will be turned off.

LIGHTS	AP/CSS
11100	Register-Immediate and Jump instructions
11100	Register-Register instructions
11100	External Register-Immediate instructions
11100	Miscellaneous instructions (LDR, SRL)
11100	Control Store Direct instructions
11100	BALR and Jump tests
11100	Local Register tests
11010	RAM test
10110	External Register tests
11000	Interrupt Switching tests
1XXX1	CSS Channel X failed
11111	Bringups Completed

X - light is flashing to indicate the failing channel
light 1 = channel 1, etc.

Figure 44. AP/CSS Bringup Errors

LIGHTS	AP/DISK
11100	Register-Immediate and Jump instructions
11100	Register-Register instructions
11100	External Register-Immediate instructions
11100	Miscellaneous instructions (LDR, SRL)
11100	Control Store Direct instructions
11100	BALR and Jump tests
11100	Local Register tests
11010	RAM test
10110	External Register tests
11000	Interrupt Switching tests
10011	Disk Adapter Communication failed
10010	Disk Controller Communication failed
0XXXX	Disk Drive X test failed
11111	Bringups Completed

X - drive number (not flashing)
light 1 = drive 1, etc.

Figure 45. AP/DISK Bringup Errors

IBM Internal Use Only

LIGHTS	AP/CSS ARIEL
1110	Register-Immediate and Jump instructions
1110	Register-Register instructions
1110	External Register-Immediate instructions
1110	Miscellaneous instructions (LDR, SRL)
1110	Control Store Direct instructions
1110	BALR and Jump tests
1110	Local Register tests
1101	RAM test
1011	External Register tests
1100	Interrupt Switching tests
1010	Control Store Cycle-Steal Indirect instructions
1010	Control Store Cycle-Steal Addressing test
XXXX	CSS Channel X failed
1001	Bringups Completed
1111	AP Initialization Completed

X - light is flashing to indicate the failing channel
light 0 = channel on interrupt level 2, etc.

Figure 46. AP/CSS ARIEL Bringup Errors

LIGHTS	AP/DIS ARIEL
1110	Register-Immediate and Jump instructions
1110	Register-Register instructions
1110	External Register-Immediate instructions
1110	Miscellaneous instructions (LDR, SRL)
1110	Control Store Direct instructions
1110	BALR and Jump tests
1110	Local Register tests
1101	RAM test
1011	External Register tests
1100	Interrupt Switching tests
1010	Control Store Cycle-Steal Indirect instructions
1010	Control Store Cycle-Steal Addressing test
1001	Bringups Completed
1111	AP Initialization Completed

Figure 47. AP/DIS ARIEL Bringup Errors

IBM Internal Use Only

10.10 COMMUNICATION TRANSMISSION SUBSYSTEM

10.10.1 Sub-System Requirements

The transmission subsystem is given messages to send by the communications manager. It transmits these messages to the specified processor, using RS-232C macrofunction cards. The highest priority messages are transmitted first, according to priorities assigned by the sender.

When a message is received for delivery to a receiving program, a low level acknowledgment is returned to the sender. If a high level response is required, it must be sent by the receiving program as another message. The line is not tied up waiting for the receiver to process a message.

The subsystem must cope with the eccentricities of the RS-232 macrofunction. Errors in transmission are automatically re-tried, up to a preset number of times.

The subsystem design must not prevent implementation of the following communications functions:

- multiple buffering by user programs,
- transferring more than one message each way between a pair of user programs,
- communication from one program to two or more other programs at once.

10.10.2 Basic Protocol

10.10.2.1 Arbitration

Each RS-232 line between processors is operated on a peer-to-peer basis. The line is idle until one side has a message to send. Either processor may begin a "session" by sending an initial bid for the line. When communications is established, the line is run in half-duplex mode: each transmission is followed by a response. Communication continues until neither side has messages to send. No special measures are required to 'start' the line, other than sending an initial bid.

The highest priority among all messages to be sent by a processor is the current bid of that processor for the line. Each transmission contains a six byte control field, giving the current bid. A transmission may also include a variable length message block provided the processor has received a valid bid, and has a message to send with the same or higher priority. The bid sent with a message is the priority of the next message to be transmitted.

10.10.2.2 Responses

Each message is assigned a 4-bit message number when it is first transmitted. When a message is successfully received, the message number is returned in the control field of the transmitted response, as an ACK number.

Any errors encountered are also indicated in the returned control field. Errors are grouped into re-triable errors, such as receive parity checks, and permanent errors, such as unknown message destinations.

The subsystem will continue attempting to send a message until

- it is successfully acknowledged,
- a permanent error is reported, or
- a predefined number of re-triable errors have been reported.

IBM Internal Use Only

Once a message has been transmitted the subsystem will not transmit a different message of any priority, until finished with the first. The current bid will still, however, reflect the highest priority message.

Successful acknowledgement of a message implies the following:

- No parity, framing, or LRC errors were detected in either the control field or the message block.
- The message destination is 'known' to the system, either as a loaded program, or (on the Series/1) as a 'loadable' system program.
- Storage was available to hold the message.
- The message was successfully unloaded into the buffer.

10.10.2.3 Response Timeout

In half-duplex, when a processor has transmitted a block, it cannot transmit again until a response has been received. To allow recovery should the other side be down, the subsystem starts a timer after each transmission. New SEND requests will not be processed until after a response is received, or the timeout expires.

The value of the timeout is on the order of 3 seconds.

10.10.2.4 Transmit Timeout

Following each transmission, the subsystem records a time when the transmission should be finished. If the transmission is not completed before the next interrupt, the subsystem waits until the recorded time. If the transmission is not complete after the recorded time, the transmit buffer is reset; this is a re-triable error. The value of the time interval is not critical; 1.2 seconds is adequate at 9.6 Kbaud.

10.10.2.5 Communication Modes

The transmission subsystem has several distinct modes of operation, depending on the traffic at each node.

Idle	No communication in progress. Either side may open communications at any time, by sending an Initial Bid.
Initial Bid	An Initial Bid has been sent by this side. The response timeout is active, and the subsystem is expecting an initial bid response. Timeout is a re-triable error.
Send	A "session" has been established, and we have a message to send (or to be acknowledged). The response timeout is active. Timeout is a re-triable error.
Receive	We have no message to send, but expect that the other side does. The response timeout is active.

10.10.2.6 Stopping and Starting

The subsystem goes into idle mode after transmitting a response if

- there are no messages to send,
- a valid 'pass' bid was received from the other side (with or without a message),
- no receive overflow was detected, and
- the transmitted error code is not a re-triable error.

IBM Internal Use Only

In this case, the no reply bit in the transmitted control field will be set on to indicate the end of communication.

Because the DSC is substantially faster than the Series/1, it could overrun the Series/1 receive buffer with a new initial bid. To reduce the problem, the DSC sets a short timeout after its last transmission, and enters receive mode. This timeout (the receive idle time) is on the order of 20 milliseconds.

The subsystem goes into idle mode without transmitting a response if there are no messages to send, and

- no response was received (response timeout), or
- a valid control field was received with the no reply bit on, and
- no receive overflow.

If we have a message to send, an initial bid is required:

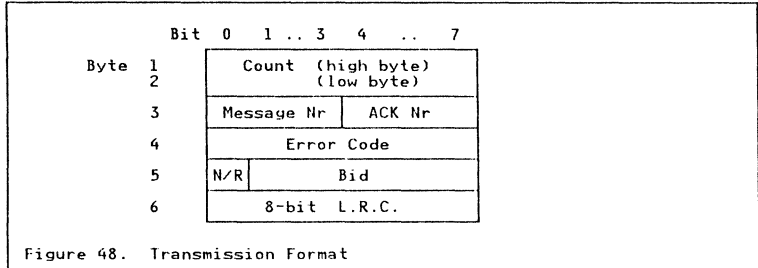
- when in idle mode (new message),
- following a response timeout (unless blast reset is sent), or
- when a valid control field is received with the no reply bit on.

The response to a valid initial bid will be flagged as a special 'bid response'. It may or may not include a message, depending on bidding.

10.10.3 Transmission Format

10.10.3.1 Control Format

The control field is the first (or only) 6 bytes of the transmitted block.



Count The two count bytes hold the byte count code passed to the RS-232 card. This is a 10-bit code, giving the number of bytes in the transmitted block (including the count bytes), minus one, arranged as follows:

Byte 1	0	1	2	3	4	-	-	-
Byte 2	5	6	7	8	9	-	-	-

('-' represents "don't care".)

e.g. 6 byte Controls + 200 byte Message Block = 206 bytes
 206 - 1 = '00 1100 1101'B
 Count Bytes = '0011 0--- 0110 1---'B
 = X'3068'

IBM Internal Use Only

Message Nr The serial number of the transmitted message (if any).

ACK Nr The serial number of the last message successfully received.

Error Code The error code for the last transmission received. The following codes are used to identify particular transmissions:

- X'FF' - Initial Bid.
- X'01' - Response to initial bid (no errors),
- X'00' - Last transmission received without errors.

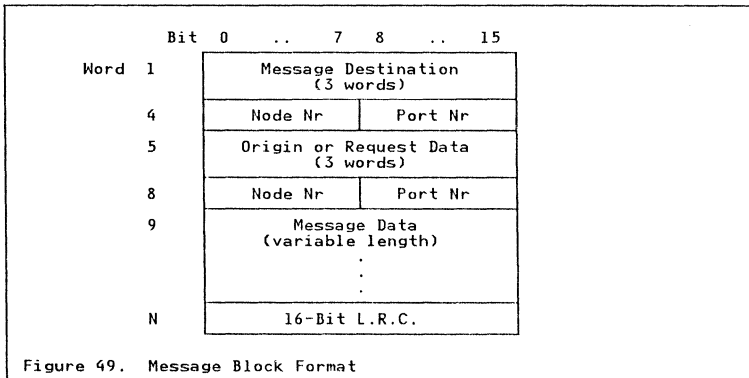
N/R The no reply bit; set to 1 if the subsystem does not expect a response from the other side.

Bid The priority of the next message to be transmitted. Priorities are between 0 (highest) and 126 (lowest). A bid of 127 is known as a pass bid, and indicates that there are no messages to be transmitted.

LRC One byte Longitudinal Redundancy Check; the exclusive OR of the first 5 bytes of the control field.

10.10.3.2 Message Block Format

The message header format is primarily determined by the high level communications management. The message size is computed from the count bytes in the control field, and may be between 0 and 500 inclusive.



10.10.4 Awkward Situations

10.10.4.1 Tie Conditions

A tie condition occurs when both sides transmit before either side receives. In the current protocol, this is possible:

- when the line is idle, and both sides send initial bids
- following a response timeout, when one or both sides send an initial bid, or
- if the final transmission in a session is not received correctly, and the side sending it attempts an initial bid.

IBM Internal Use Only

(The 'tie window' for tied initial bids is roughly the time required to transmit the six byte bid: 7 msec. at 9.6 Kbaud.),

To prevent loss of synchronization in the protocol, the subsystem must be able to discard particular initial bids. The discarding side sends no response to the initial bid; it waits for a response from its previous transmission (which 'tied' with the initial bid).

10.10.4.2 Receive Overflow

Receive overflow in the RS-232C macrofunction occurs when a transmission begins before the previous transmission is cleared from the receive buffer. The second transmission is lost. Receive overflow is possible:

- if one side sends an initial bid immediately after sending the final transmission in a session, or
- after a tie condition, when the responding side transmits before its initial bid has been discarded.

The received transmission would not normally produce a response. The receiving side must recognize the receive overflow condition, and transmit a response.

If the old data is cleared from the buffer while an overflow block is being received, then the remainder of the overflow block will enter the buffer, possibly leading to a partially full receive buffer. To avoid this, the overflow message must be as short as possible. Thus, the response after a tie condition should be restricted to a control field only.

10.10.4.3 Handling Tie Conditions and Receive Overflow

The subsystem sets a 'Discard' flag as part of each processing pass. The flag is set off if

- we are entering idle mode,
- a valid control field is received with the 'no reply' bit off,
- the last received block was discarded, or a complete 'blast reset' is received.

The latter three conditions indicate the other side has set its response timeout, and will not transmit until the timeout expires. If the timeout is not active, the other side could transmit an initial bid at any time.

An initial bid received while a transmission is in progress, or when the discard flag is on, is a candidate for discarding. If the last transmission was not an initial bid, the received bid is always discarded. If both sides have sent initial bids, the lower priority bid is discarded; if the bids are equal, the Series/1 discards the bid it receives. This is the only point in the subsystem in which the two ends of the line are not equivalent.

The side sending the discarded initial bid will receive a prior transmission from the discarding side. This transmission will not be a 'bid response'. To avoid receive overflow, only a control field is sent when the response to an initial bid is not a 'bid response'. (No message is sent, regardless of the bid received.)

10.10.4.4 Partially Full Receive Buffer

The RS-232C macrofunction has a partially full receive buffer when there is data in the buffer with the byte count unsatisfied, and no transmission is in progress. This can result from

- a transmission error in the byte count,
- loss of transmitted data,

IBM Internal Use Only

- stoppage of transmission from the other side, or
- receive overflow, as described earlier.

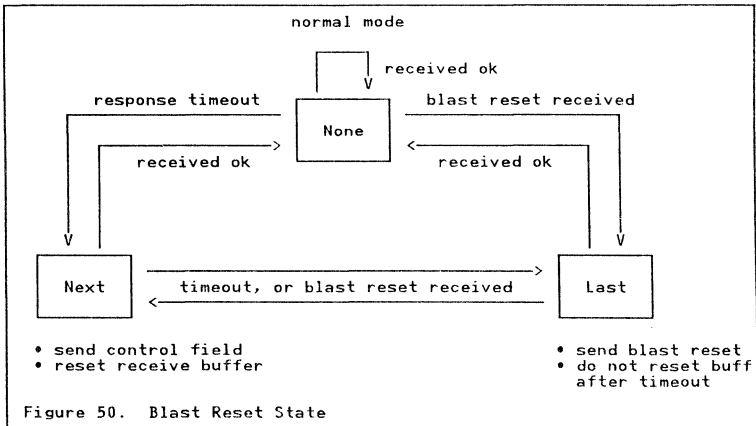
This cannot be detected with certainty from either side. The receiving side, can recover by resetting its receive buffer, provided no transmission is in progress. The transmitting side can send enough data to fill the buffer; excess data will overflow provided the buffer is not cleared while the transmission is in progress. The receiver cannot tell when a transmission is in progress.

10.10.4.5 Handling Partially Full Receive Buffers

The transmitting side uses a distinctive sequence of 1024 bytes of data in attempting to fill the partially full buffer. This sequence is known as a Blast Reset.

The subsystem assumes that the other side has a partially full buffer when the response timeout fails twice in succession. It also assumes this if a complete blast reset is received, indicating that the other side is not receiving responses.

The subsystem maintains a 'Blast Reset State' to determine when to send a blast reset.

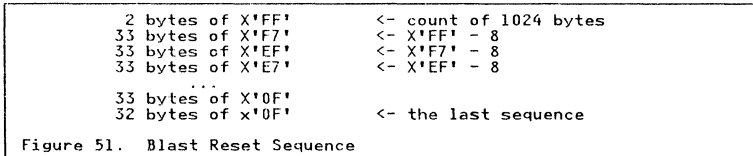


Transmitting 1024 bytes of data will always generate at least one receive interrupt. If the other receive buffer is partially full, sending a blast reset will cause the other side to send an error response. If our buffer is partially full, sending a blast reset will cause the other side to return a blast reset. If no response is received to a blast reset, then either communication is down, or both receive buffers are partially full. In this case, the subsystem assumes communication is down; an error code is returned to the outstanding SEND command, if any, regardless of the re-try count.

IBM Internal Use Only

10.10.4.6 The Blast Reset Format

If the receive buffer is reset while a blast reset is incoming, the next two bytes received will be interpreted as a new byte count. Because of this, the blast reset sequence is structured as shown in the figure below. Note that, the bytes are in descending order and the low order three bits in each byte are all on.



This sequence has the property that any two consecutive bytes, interpreted as a byte count, are within 32 of the number of bytes remaining in the transmission. Except for the last 33 bytes, the count will always be less than the length remaining. Hopefully, the excess bytes will overflow before the buffer is again reset. In the worst case, the buffer will still be partially full, with a byte count of 34 to be satisfied.

The low order bits of a normal byte count are always zero, and the count is always even. If a count of X'FFFF' is received, then we assume a complete blast reset has arrived. If the low order bits of both count bytes are all ones, and the bytes differ by X'00' or X'08', the transmission is assumed to be a 'fragment' of a blast reset. All such fragments are discarded. Any other combination in the count bytes is reported to the other side as a bad count field.

10.10.5 Interface to Communications Management

10.10.5.1 Send

The SEND support adds valid SEND commands to a queue in order of priority. If the subsystem is idle when a SEND is queued, it will be posted by communications management.

As each SEND is processed, the subsystem removes the command from the queue, and posts the user with a return code.

10.10.5.2 Receive

When a message arrives, the subsystem calls entry point 'FINDBUFF' in the communications manager. It supplies the destination and origin information, and the message size. FINDBUFF verifies that the destination is valid and a place is available to receive the data, and supplies an appropriate return code.

If the message is acceptable, FINDBUFF returns the location where the message is to be stored. The subsystem unloads the message data into the buffer provided. The 'RETNBUFF' entry point is used to report whether the operation was successful, and to return the buffer for delivery (or return to the buffer pool).

IBM Internal Use Only

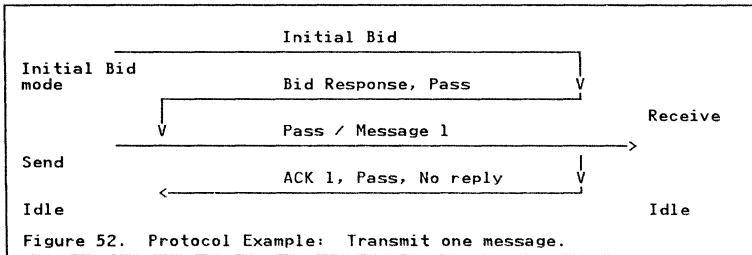
10.10.6 Primary Entries in the Line Control Block

The line control block contains the data specific to an RS-232 line that must be retained between processing cycles of the subsystem (between interrupts). It provides the interface between SEND command processing, and the subsystem.

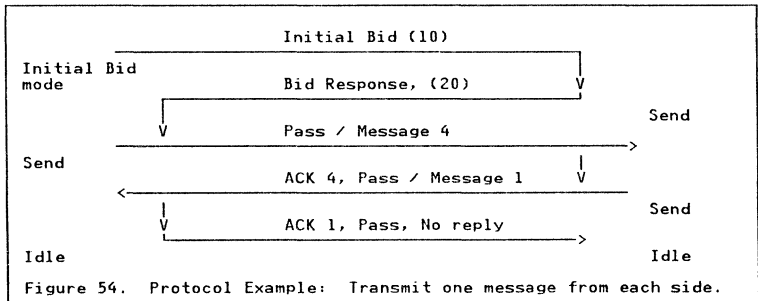
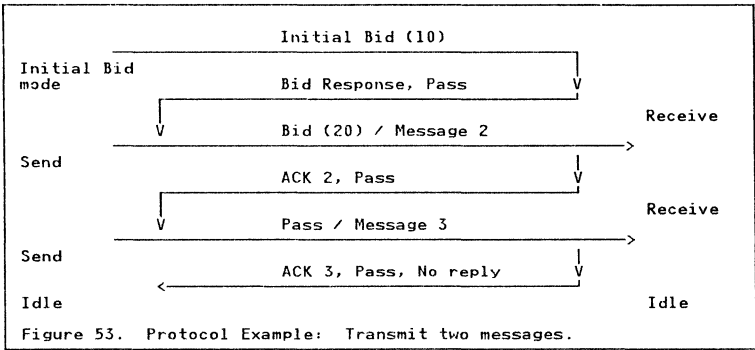
Primary entries in the Line Control Block are:

- Current Communication Mode
- Our Current Bid
- Pointer to Current SEND Command (0 = none)
- Serial Number of Current Message
- ACK number to be sent to other side
- Error re-try counter (number of additional re-triable errors before the current message is abandoned)
- Discard flag
- Blast reset state
- Expected completion time for last transmission
- Priority of first message in SEND queue
- SEND command queue (highest priority first)

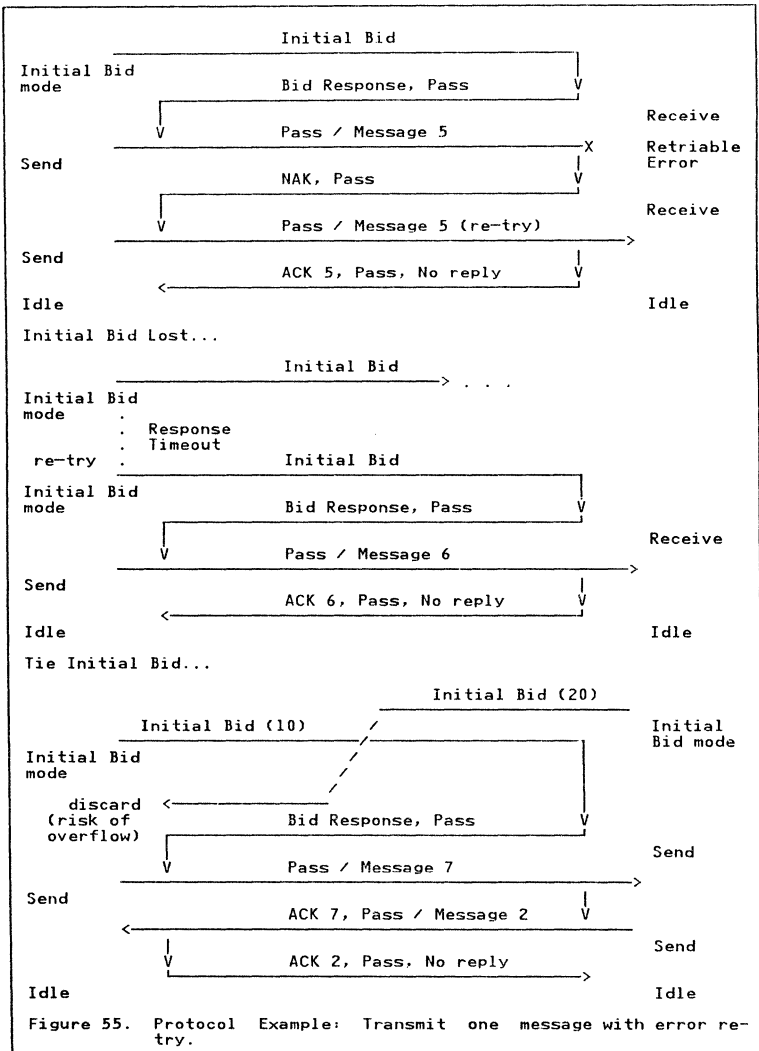
10.10.7 Protocol Examples



IBM Internal Use Only



IBM Internal Use Only



IBM Internal Use Only

10.10.8 Program Loader

The LOADH instruction loads a program from the Series/1 Host to the DSC. The following shows the macro layout of the LOADH instruction:

LOADH MACRO LAYOUT

Word	Description
1	OP CODE
2	Flag
3-6	EBCDIC program name
7	LOADPOINT address
8	ERROR address
9	Parameter list address
10	ECB address
11	ECB key

FLAGS LAYOUT

Bit	Description
0	PARMLIST was specified
1	EXEC (ATTACH) = YES
2	LOGMSG = YES
3	ERROR = SPECIFIED
4	EVENT = SPECIFIED
5	NUCLEUS load
6	DATA Load

Return Code

Code	Description
0	Successful
1	Invalid Program name
2	Program name not found on disk
3	Invalid segment number
4	Not a tool program
5	Disk error in getting the program
6	Segment overrun
7	Host communications error
8	Insufficient space for loading
9	LSA not available
A	Program length (in the header) = 0

10.10.8.1 Request Block

EDX/J

EDX/J4

1)	A000 REQUEST	EBCDIC PROGRAM NAME (8 BYTES)
2)	DATA LENGTH ON WRITE REQUEST (usually 768 bytes)	
3)	FLAGS (SYSTEM RESERVED)	
4)	EBCDIC PROGRAM NAME (8 BYTES)	
8)	PROGRAM SEGMENT NO. REQUESTED 0 = HEADER INFO REQUESTED -0 = PROGRAM INFO REQUESTED	

IBM Internal Use Only

10.10.8.2 Header Response Block

EDX/J4:	\$HLCISZ	DATA	X'0000'	Core Image Data Size (bytes)
	\$HLRLDS	DATA	X'0000'	RLD Size (words)
	\$HLFOLA	DATA	X'0000'	Program Load
	\$HLPOEN	DATA	X'0000'	Program Entry Point (not used)
	\$HLTOLS	DATA	X'0000'	Total Number of Segments (TXT + RLD)
	\$HLBKSZ	DATA	X'0000'	Block Size Requested
EDX/J4:	HEADBUFF	EQU	*	
	RTNCODE	DATA	F'0'	Initial Load Return Code
	RTNAME	DATA	4F'0'	Official Load Name
	TXTRC#	DATA	F'0'	Program Text Record Size
	RLDSIZ	DATA	F'0'	RLD Size (words)
	LOADPT	DATA	F'0'	Program Load Point
	RTNDEST	DATA	4F'0'	Part2 LOADER Destination
	#HDREND	EQU	*	
	#HDRELEN	EQU	(#HDREND-HEADBUFF)/2	

10.10.8.3 LOAD Sequence:

1. Allocate a contiguous memory region to contain the program or data. Go to ERROR handler if such region is available.
2. Receive segments, if necessary, of both text and RLDs from the HOST and place at determined location. If a DATA load has been requested then no RLDs, IDBs or MDBs are loaded.
3. Relocate program addresses using the RELOCATION DICTIONARY (RLD) loaded with the text.
4. IDBs are linked into the IDB Pointer Table. A check is done to ensure that the IDB has not been previously defined and if it has been, the loader aborts. No such table exists and no checking is done for MDBs.
5. Perform a PROGRAM LEVEL CHECK to ensure that the program is capable of running on this particular nucleus.
6. Place the address of SYSCOM into the PROGRAM HEADER
7. Place the program load point into the #TCBPLP field of all the program TCBS, and the KEY in the #TCBADS field of all program TCBS.
8. Print the LOG message if requested.
9. Move the size of the parameter list and the list itself (if one exists) to the end of the PROGRAM HEADER. (optional)
10. Move the address of the END ECB to the #PRGEECB field of the PROGRAM HEADER.
11. ATTACH the main TCB if the EXEC option was selected.
12. Place the program load point and key into word specified by the LOADPT= keyword (optional). For a DATA load, a third parameter, the size, is recorded. It is the responsibility of the user to ensure that the amount of space required to save these quantities has been allocated.

Note:

1. Program text and RLD data will always be loaded at the start of 256 word page (ie. low address byte equals 0) and any space unused in the page will be set to zero.
2. The RLD is a list of the addresses in the program that must be relocated. Relocation is accomplished by adding the program load point to the referenced location. It was mentioned in the section describing the USER command that certain JLB instructions should not be used in relocated programs because their operands are not relocatable. This occurs because of the fact that the address is encoded into the command at the time of assembly not loading.

IBM Internal Use Only

10.10.9 Layout of the Communication Macros

OPEN MACRO LAYOUT

Word	Description
1	OP Code
2	Response OCB Chain (from Big Table)
3-5	Name (3 WORDS)
6	Send Chain (Address of Head of Close Chain)
7	Open Flags
8	Port 0 Pointer
9	Number of Ports
10	Port Control Block

SEND MACRO LAYOUT

Word	Description
1	Flag/OP Code
2	Destination (or Dest displacement)
3	Buffer (or Buffer displacement)
4	Word Count
5	Active Chain Link (msg queue for this line)
6	Priority
7-10	Save area for Destination, Buffer & Count (3 words)
11	Return Code
12	OCB Address
13	Response Port or Data Address
14	End Event (or address of TCB.ECB)
15	Preceding Send (Static Close Chain)
16	Close Flags

FLAG LAYOUT

Bit	Description
0	not used
1	= 1 if EVENT= not specified (wait on TCB.ECB)
2	= 1 if DATA= was specified
3	not used
4,5	Buffer index register
6,7	Destination index register

RECEIVE MACRO LAYOUT

Word	Description
1	STIMER (Start STIMER without wait if TIME= present)
2	Timeout
1	Flags/Opcode
2	Buffer (or Buffer displacement)
3	Word Count
4	Save area for Buffer & Count (2 words)
6	OCB Address
7	Receive Port
8	Return Code
9	Nowork Address
10	Wait Event (TCB.ECB or TCB Timer ECB)
11-14	Origin Field (4 words)
15	Received Count

FLAG LAYOUT

Word	Description
0	= 1
1	= 1 if NOWORK= specified (no wait)
2	= 1 if TIME= specified (wait on TCB Timer ECB)
3,4	not used
6,7	Buffer index register

IBM Internal Use Only

10.11 DSC IPL LOAD

10.11.1 IPL Loader in Hazel

1. An IPL NUCLEUS is transferred from ROS to RAM at location X'0000'.
2. The IPL NUCLEUS loads from the HOST the user specified nucleus starting at location x'4000'.
3. The IPL NUCLEUS builds in high memory a NUCLEUS MOVER to which it then branches.
4. The NUCLEUS MOVER moves the new nucleus to location x'0000', re-initializes the machine. The new nucleus then starts to run in a the normal manner.

Note:

This IPL procedure does not apply to the use of a regular ROS NUCLEUS from a different ROS card.

10.11.2 IPL Loader in APs

Hazel/4 may be IPLed from EPROM card, AP/CSS or AP/DISK. The IPL device is hardware selectable. The IPL sequence defines what is the back up IPL device in case the primary IPL device fails. The IPL sequence is:

1. EPROM card.
2. AP/CSS - from S/1.
3. AP/DISK - from hard or floppy disk.

The AP/CSS in Ariel System does not have IPL sequence. Ariel AP/CSS has two sources of IPL: S/1 and EPROM. Switch "0" on the AP/CSS card determines the source of IPL:

1. Switch "0" = ON - EPROM IPL
2. Switch "0" = OFF - S/1 IPL

10.11.2.1 IPL Loader in AP/DISK

The name of the nucleus loaded by AP/DISK is \$EDXIPL. The nucleus may be loaded from a floppy or a hard disk.

1. After completing bringups AP/DISK resets Hazel and holds it in the reset state to prevent JIB from executing while nucleus is being loaded.
2. After the IPL load is completed AP/DISK releases Hazel from the reset mode and Hazel starts its bringups.

Note:

1. Communication errors may occur during the IPL load in which case AP/CSS will display error message on the 3101 and restart the IPL load.
2. The floppy disk has a priority over the hard disk.
3. AP/DISK does not display any messages on 3101 while ipling.

IBM Internal Use Only

10.11.2.2 IPL Loader in AP/CSS

The name of the nucleus loaded by AP/CSS is TxNC3 (where xx=99 or the TP line number).

1. After completing bringups AP/CSS resets Hazel and holds it in the reset state to prevent JIB from executing while nucleus is being loaded.
2. Next, AP/CSS displays 'IPLING DSC...' message on 3101.
3. As the the nucleus is being loaded to Hazel's memory AP/CSS displays on the 3101 the number of the nucleus records to be transferred.
4. After the IPL load is completed AP/CSS releases Hazel from the reset mode and Hazel starts its bringups.

Note:

1. Communication errors may occur during the IPL load in which case AP/CSS will display error message on the 3101 and restart the IPL load.
2. The error message format is:

 RC = rc1 rc2 rc3 rc4 rc5

 rc1 - 0000 = time-out, message was not received.
 - 80F0 = message received.
 - else = hardware transmit or receive error.

 rc5 - TP error code. See "SEND" on page 148.
3. A more detailed explanation of the AP/CSS IPL communication errors can be found in the AP/CSS program listing.
4. If IPL load fails AP/CSS will enter debug and hold Hazel in the reset state.

IBM Internal Use Only

IBM Internal Use Only

APPENDIX A. SYNTAX SUMMARY

Commands	Operands	
-----	-----	
ADD	opnd1,opnd2,count,RESULT=,PREC=,P1=,P2=,P3=	
ADDV	opnd1,opnd2,count,RESULT=,PREC=,P1=,P2=,P3=	
AND	opnd1,opnd2,count,RESULT=,PREC=,P1=,P2=,P3=	
APU		
ATTACH	taskname,priority,CODE=,KEY=,P1=,P2=,P3=,P4=	
ATTNLIST	(cc1,loc1,,ccn,locn)	
BUFFER	count,INDEX=name	
CALL	name,param1, . . . ,param5,P1=, . . . ,P6=	
CANCEL	loc,KEY=,P1=,P2=	
CLOSE	STOP=,LOGMSG=	
COMM	ADDRESS=,INTBIT=	
CONVTB	opnd1,opnd2,PREC=,FORMAT=,P1,P2	
CONVTD	opnd1,opnd2,PREC=,FORMAT=,P1,P2	
CPU	CARDS=	
DATA	expression	
DEQ	resource,code,KEY=,P1=,P2=,P3=	
DEQT		
DEST	name,port,NODE=,P2=	
DETACH	code,P1=	
DISIO	mdcb,EVENT=,ERROR=,P1=,P2=	ARIEL
DISK	TYPE=IBM	
DISREAD	LSAx,LOC,CTC=,BITS=(u,v),COUNT=,ERROR=,P1=,P2=	
DISREAD	LSAx,loc,CTC=,BITS=(u,v),COUNT=,ERROR=,EVENT=, P1=,P2=,P5=,P6=	ARIEL
DISWRITE	LSAx,LOC,CTC=,BITS=(u,v,rctc),COUNT=,ERROR=,P1=,P2=	
DISWRITE	LSAx,loc,CTC=,BITS=(u,v,rctc),COUNT=,ERROR=,EVENT=, P1=,P2=,P5=,P6=	ARIEL
DIVIDE	opnd1,opnd2,count,RESULT=,PREC=,P1=,P2=,P3=	
DO	count,TIMES,INDEX=,P1=	
DSCB	dsname,volname,P1=,P2=	
ECB	code	
EJECT		
ELSE		
END		
ENDATTN		
ENDDDO		
ENDIF		
ENDPROG		
ENDSYS		
ENDTASK	code,P1=	
ENDTP		
ENQ	resource,BUSY=,KEY=,P1=,P2=,P3=	
ENQT	name,BUSY=	
EOR	opnd1,opnd2,count,RESULT=,PREC=,P1=,P2=,P3=	
EXP	opnd1,opnd2,P1=,P2=	
FADD	opnd1,opnd2,RESULT=,P1=,P2=,P3=	
FDIVD	opnd1,opnd2,RESULT=,P1=,P2=,P3=	
FMULT	opnd1,opnd2,RESULT=,P1=,P2=,P3=	
FPCONV	opnd1,opnd2,COUNT,PREC=,P1=,P2=,P3=	
FREEMAIN	loadpoint,KEY=,DATA,ERROR=,P1=,P2=,P3=	
FSUB	opnd1,opnd2,RESULT=,P1=,P2=,P3=	
GETMAIN	pagecount,location,ERROR=,KEY=,P1=,P2=,P3=,P4=	
GETTIME	loc,DATE=,P1=	
GETVALUE	loc,msg,count,MODE=,PROMPT=,FORMAT=,TYPE=, SKIP=,LINE=,SPACES=,COL=,P1=,P2=,P3=	
GOTO	loc,P1=	
GOTO	(loc0,loc1,loc2, . . . ,locn),index,P1=,P2=	
HDWRTEST	LRCHECK=	
IF	statement<,THEN>	
IF	statement,GOTO,loc	
INTBIT	INTx,BLOCK=,BIT=,ECB=	
INTIME	reftime,loc,index,P2=	

IBM Internal Use Only

IOCB	PASIZE=,OVFLINE=,NHIST=	
IOR	opnd1,opnd2,count,RESULT=,PREC=,P1=,P2=,P3=	
LN	opnd1,opnd2,P1=,P2=	
LOG	opnd1,opnd2,P1=,P2=	
LOAD	dscb,MODE=,LOADPT=,ERROR=,PARMNAM=, EVENT=,KEY=,LOGMSG=,P1=,P2=,P3=	
LOADH	pgmname,PARMNAM=,(NO)EXEC,DATA,LOADPT=,LOGMSG=, ERROR=,EVENT=,LOADNUC=,KEY=,P1=,P2=,P3=	
LSA	LSAx,TYPE=,BLOCK=,MACRO=,BITS=(u,v)	
LSA	LSAx,BLOCK=,MACRO=,LSA=(block,macro),TYPE=	ARIEL
MAPAD	loc,P1=	
MDCB	cmd,buf,LSA=,CTC=,COUNT=,BITS=,TYPE=,RESET=,RETRY=, CHAIN,P2=,P3=,P4=,P5=,P6=,P7=,P8=	ARIEL
MOVE	opnd1,opnd2,count,FKEY=,TKEY=,P1=,P2=,P3=	
MOVEA	opnd1,opnd2,P1=,P2=	
MULTIPLY	opnd1,opnd2,count,RESULT=,PREC=,P1=,P2=,P3=	
NUCTYPE	ENVIR=	
OPEN	name,PORTS=	
POST	event,code,KEY=,P1=,P2=,P3=	
PRINDATE		
PRINT	ON/OFF,GEN/NOGEN,DATA/NODATA	
PRINTTEXT	msg,LINE=,SKIP=,SPACES=,COL=,MODE=,P1=	
PRINTIME		
PRINTNUM	loc,count,nline,nspace,MODE=,FORMAT=,TYPE=, SKIP=,LINE=,SPACES=,P1=,P2=,P3=,P4=	
PROGRAM	start,priority,EVENT=,PARM=n,TERMERR=	
PROGSTOP	code,LOGMSG=YES/NO,P1=	
PWR	opnd1,opnd2,P1=,P2=	
QCB	code	
QUESTION	pmsg,YES=,NO=,P1=	
READ	dscb,loc,recnt,relrec,KEY=,WAIT=,END=, ERROR=,PREC=,P1=,P2=,P3=,P4=,P5=	
READTEXT	loc,pmsg,PROMPT=,PROTECT=,MODE=,P1=,P2=	
RECEIVE	buffer,count,PORT=,TIME=,NOWORK=,RCNT=,ORG=, KEY=,RC=,P1=,P2=,P3=,P4=	
RESET	event,KEY=,P1=,P2=	
RETURN		
SEND	dest,buffer,count,EVENT=,PRIOR=,PORT=,DATA=,RC=, KEY=,P1=,P2=,P3=,P4=	
SHIFTL	opnd1,opnd2,count,RESULT=,PREC=,P1=,P2=,P3=	
SHIFTR	opnd1,opnd2,COUNT,RESULT=,PREC=,P1=,P2=,P3=	
SPACE	value	
SQRT	opnd1,opnd2,P1=,P2=	
STIMER	count,WAIT,P1=	
SUBROUT	name,par1,...,par5	
SUBTRACT	opnd1,opnd2,count,RESULT=,PREC=,P1=,P2=,P3=	
SYSCDEF	disp,(type,count,value/address), (type,count,value/address)	
SYSCOPEN	syscdef	
SYSTEM	STORAGE=,MAXPROG=	
TASK	start,priority,EVENT=,TERMERR=	
TCBGET	loc,tcbfld,P1=	
TERMINAL	DEVICE=,RAM=,VIDEO=,PAGSIZE=,LINESIZE=, LINEDEL=,CHARDEL=,CRDELAY=,ECHO=,NHIST=,OVFLINE=	
TEXT	'message',LENGTH=	
TIME	loc,P1=	
TITLE	'message'	
USER	name,PARM=(parml,...,parmn),P=(namel,...,namen)	
WAIT	event,RESET,TIMEOUT=,KEY=,P1=,P2=,P3=	
WHERESES	progrname,address,KEY=,P1=,P2=,P3=	
WRITE	dscb,loc,recnt,relrec,KEY=,WAIT=,END=, ERROR=,PREC=,P1=,P2=,P3=,P4=,P5=	
\$SYSCOM	COMSIZE=	

IBM Internal Use Only

APPENDIX B. EQUATE TABLES

The following is a listing of the equated labels which may be used to refer to specific fields within various System Control blocks.

B.1 NUCLEUS DIRECTORY AND EQUATES

At the front of the nucleus is a table that contains pointers to various data structures used throughout the system. This table is known as the Nucleus Directory and can be referenced by the equate #NUCDIR which contains a pointer to the location \$NUCDIRA. This location contains the address of the nucleus directory NUCDIRC.

The following example shows how to retrieve the READY QUEUE pointer from a Nucleus Directory:

```

MOVE    #1,0
MOVE    #1,(+ #NUCDIRA,#1)      | #1 <- address of NUCDIRC
MOVE    READYQ,(+ #NCRDYQ,#1)  | READYQ <- A($READYQ)

```

The contents of Nucleus Directories and the associated equates vary depending on the type of nucleus (e.g. Hazel/3, Hazel/4 or Hazel/Ariel).

B.1.1 Nucleus Directory in Hazel/3

NUCDIRC	DATA	A(\$EXEC)	50	EDX S/W STOP ADDRESS
	DATA	A(\$MDTRACE)	51	CURRENT TRACE ADDRESS
	DATA	A(\$BRINGUP)	52	TRACE START ADDRESS
	DC	H'256'	53	TRACE SIZE
	DATA	A(\$READYQ)	54	READYQ POINTER
	DATA	A(\$TIMCHAN)	55	INTERVAL TIMER QUEUE
	DATA	A(\$TIMEND)	56	LVLI SUPPORT ADDR HI
	DATA	A(\$REGMAP)	57	REGISTER / PSW DUMP
	DATA	A(\$LINE01)	58	COMMUNICATIONS TASK
	DATA	A(\$LOAD)	59	LOADER TASK
	DATA	A(\$UPUTL)	5A	ATTENTION TASK
	DC	H'0'	5B	SPARE
	DATA	A(LCB)	5C	LCB POINTER
COMTPTR	DC	X'D000'	5D	COMM TRACE POINTER
\$NUCLVL	DC	H'2'	5E	NUCLEUS LEVEL = 2
	DC	4H'0'	5F/62	UNIQUE TO h3/AP
	DATA	A(\$SYSLOG)	63	CCB
	DATA	A(\$CNCLQCB)	64	CANCEL QCB
\$CSCPTR	DC	H'0'	65	CSCP ADDRESS
\$ESFPTR	DC	H'0'	66	\$ESFPTRA ADDRESS
	DC	2H'0'	67/68	IBM DISK
	DC	2H'0'	69/6A	IMC BASE0,KEY
	DC	3H'0'	6B/6D	NUCLEUS SVC ENTRY PTS

Figure 56. Nucleus Directory in Hazel/3

#NCEXEC	EQU	0	START OF INTERPRETER MAINLINE (TESTER)
#NCMDTP	EQU	1	MINIDEBUG CURRENT EDX TRACE POINTER
#NCMDTA	EQU	2	MINIDEBUG EDX TRACE AREA
#NCMDTS	EQU	3	MINIDEBUG EDX TRACE AREA SIZE
#NCRDYQ	EQU	4	SUPERVISOR READY Q
#NCLVLI	EQU	5	START ADDRESS FOR LEVEL 1 TIMER SUPPORT
#NCLVLI1	EQU	6	END ADDRESS FOR LEVEL 1 TIMER SUPPORT
#NCRGMAP	EQU	7	START ADDRESS FOR REGISTER/PSW DUMP MAP
#NCLINE1	EQU	8	COMMUNICATIONS TCB ADDRESS
#NCLNLDLDR	EQU	9	UTILITY LOADER TCB ADDRESS
#NCUPUTL	EQU	10	UTILITY ATTENTION TCB ADDRESS

IBM Internal Use Only

*#NCLCB	EQU	11	SPARE
*#NCCOMTR	EQU	12	COMMUNICATIONS LCB.ERROR LIST POINTER
*#NCLVL	EQU	13	COMMUNICATIONS TRACE POINTER
*#NCLVL4	EQU	14	NUCLEUS LEVEL
*#NCPF	EQU	15	HAZEL4/AP INTERFACE TABLES
*#NCCCB	EQU	16	HAZEL4/AP PF TABLE
*#NCCNCL	EQU	17	SPARE
*#NCCSCP	EQU	18	SPARE
*#NCESFP	EQU	19	CCB
*#NCDISK	EQU	20	CANCEL QCB
*#NCDISKO	EQU	21	CSCP
*#NCIMC	EQU	22	ESF
*#NCIMCK	EQU	23	NEW DISK WORK QUEUE
*#NCENQ	EQU	24	OLD DISK WORK QUEUE
*#NCDEQ	EQU	25	INTELLIGENT MACROFUNCTION CARD BASED
*#NCHAITO	EQU	26	INTELLIGENT MACROFUNCTION CARD KEY
*#NCTSKTR	EQU	27	SUPERVISOR ENQ ENTRY POINT
		28	SUPERVISOR DEQ ENTRY POINT
		29	INTERNAL WAIT/TIMEOUT ENTRY POINT
		30	TASK TRACE TABLE @

B.1.2 Nucleus Directory in Hazel/4 without AP/CSS

NUCDIRC	DATA	A(\$EXEC)	50	EDX S/W STOP ADDRESS
	DATA	A(\$MDTRACE)	51	CURRENT TRACE ADDRESS
	DATA	A(\$BRINGUP)	52	TRACE START ADDRESS
	DC	H'256'	53	TRACE SIZE
	DATA	A(\$READYQ)	54	READYQ POINTER
	DATA	A(\$TMICHAN)	55	INTERVAL TIMER QUEUE
	DATA	A(\$TIMEND)	56	LVL1 SUPPORT ADDR HI
	DATA	A(\$REGMAP)	57	REGISTER / PSW DUMP
	DATA	A(\$LINE01)	58	COMMUNICATIONS TASK
	DATA	A(\$LNLOAD)	59	LOADER TASK
	DATA	A(\$SVUTIL)	5A	ATTENTION TASK
	DC	H'0'	5B	SPARE
	DATA	A(\$LCB)	5C	LCB POINTER
COMTPTR	DC	X'A000'	5D	COMM TRACE POINTER
\$NUCLVL	DC	H'2'	5E	NUCLEUS LEVEL = 2
	DC	4H'0'	5F/62	UNIQUE TO h3/AP
	DATA	A(\$SYSLOG)	63	CCB
	DATA	A(\$CNCLQCB)	64	CANCEL QCB
\$CSCPTR	DC	H'0'	65	CSCP ADDRESS
\$ESFPTR	DC	H'0'	66	ESFPTRA ADDRESS
	DC	2H'0'	67/68	IBM DISK
	DC	2H'0'	69/6A	AP IMC BASE@,KEY
	DC	3H'0'	6B/6D	NUCLEUS SVC ENTRY PTS
	DATA	A(\$TSKTRCT)	6E	TASK TRACE TABLE @

Figure 57. Nucleus Directory in Hazel/4 without AP/CSS

#NCEXEC	EQU	0	START OF INTERPRETER MAINLINE (TESTER)
#NCMDTP	EQU	1	MINIDEBUG CURRENT EDX TRACE POINTER
#NCMDTA	EQU	2	MINIDEBUG EDX TRACE AREA
#NCMDTS	EQU	3	MINIDEBUG EDX TRACE AREA SIZE
#NCRDYQ	EQU	4	SUPERVISOR READY Q
#NCLVL1	EQU	5	START ADDRESS FOR LEVEL 1 TIMER SUPPORT
#NCLVL1E	EQU	6	END ADDRESS FOR LEVEL 1 TIMER SUPPORT
#NCRGMAP	EQU	7	START ADDRESS FOR REGISTER/PSW DUMP MAP
#NCLINE1	EQU	8	COMMUNICATIONS TCB ADDRESS
#NCLNLDR	EQU	9	UTILITY LOADER TCB ADDRESS
#NCUPUTL	EQU	10	UTILITY ATTENTION TCB ADDRESS
*	EQU	11	SPARE
#NCLCB	EQU	12	COMMUNICATIONS LCB.ERROR LIST POINTER
#NCCOMTR	EQU	13	COMMUNICATIONS TRACE POINTER
#NCLVL	EQU	14	NUCLEUS LEVEL
#NCLVL4	EQU	15	HAZEL4/AP INTERFACE TABLES
#NCPF	EQU	16	HAZEL4/AP PF TABLE

IBM Internal Use Only

*	EQU	17	SPARE
*	EQU	18	SPARE
#NCCCB	EQU	19	CCB
#NCCNCL	EQU	20	CANCEL QCB
#NCCSCP	EQU	21	CSCP
#NCESFP	EQU	22	ESF
#NCDISKN	EQU	23	NEW DISK WORK QUEUE
#NCDISKO	EQU	24	OLD DISK WORK QUEUE
#NCIMC	EQU	25	INTELLIGENT MACROFUNCTION CARD BASE@
#NCIMCK	EQU	26	INTELLIGENT MACROFUNCTION CARD KEY
#NCENQ	EQU	27	SUPERVISOR ENQ ENTRY POINT
#NCDEQ	EQU	28	SUPERVISOR DEQ ENTRY POINT
#NCWAITO	EQU	29	INTERNAL WAIT/TIMEOUT ENTRY POINT
#NCTSKTR	EQU	30	TASK TRACE TABLE @

B.1.3 Nucleus Directory in Hazel/4 with AP/CSS

NUCDIRC	DATA	A(\$EXEC)	50	EDX S/W STOP ADDRESS
	DATA	A(\$TRPTR)	51	CURRENT TRACE ADDRESS
	DATA	A(\$BRINGUP)	52	TRACE START ADDRESS
	DC	H'256'	53	TRACE SIZE
	DATA	A(\$READYQ)	54	READYQ POINTER
	DATA	A(\$TM1CHAN)	55	INTERVAL TIMER QUEUE
	DATA	A(\$TIMEND)	56	LVL1 SUPPORT ADDR HI
	DATA	A(\$REGMAP)	57	REGISTER / PSW DUMP
	DATA	A(\$LINE01)	58	COMMUNICATIONS TASK
	DATA	A(\$LNLOAD)	59	LOADER TASK
	DATA	A(\$SVUTIL)	5A	ATTENTION TASK
	DC	H'0'	5B	SPARE
	DATA	A(LCB)	5C	LCB POINTER
COMTPTR	DC	X'A000'	5D	COMM TRACE POINTER
\$NUCLVL	DATA	X'0002'	5E	NUCLEUS LEVEL 2
	DC	AL2(\$APTB)	5F	HAZEL4/AP I/F TABLES
	DC	AL2(\$PFTB)	60	HAZEL4/AP PF TABLES
	DATA	A(\$EDFLAGS)	61	EDX DEBUG CONTROL BLK
	DATA	A(\$JDFLGS)	62	JIB DEBUG CONTROL BLK
	DATA	A(\$SYSLOG1)	63	CCB 1
	DATA	A(\$CNCLQCB)	64	CANCEL QCB
\$CSCPTR	DC	H'0'	65	CSCP ADDRESS
\$ESFPTR	DC	H'0'	66	ESFPTRA ADDRESS
	DC	AL2(\$NEWDISK)	67	NEW AP DISK WORK
	DC	AL2(\$OLDISK)	68	OLD AP DISK WORK
\$APIMC	DC	2H'0'	69/6A	AP IMC BASE@,KEY
	DATA	A(\$ENQ)	6B	SUPERVISOR \$ENQ
	DATA	A(\$DEQ)	6C	SUPERVISOR \$DEQ
	DATA	A(\$WAITO)	6D	INTERNAL WAIT ENTRY
	DATA	A(\$TSKTRCT)	6E	TASK TRACE TABLE @
	DATA	A(\$APCOMP)	6F	AP COMPATIBILITY TBL

Figure 58. Nucleus Directory in Hazel/4 with AP/CSS

#NCEXEC	EQU	0	START OF INTERPRETER MAINLINE (TESTER)
#NCDBTP	EQU	1	DEBUG CURRENT EDX TRACE POINTER
#NCDBTA	EQU	2	DEBUG EDX TRACE AREA
#NCDBTS	EQU	3	DEBUG EDX TRACE AREA SIZE
#NCRDYQ	EQU	4	SUPERVISOR READY Q
#NCLVL1	EQU	5	START ADDRESS FOR LEVEL 1 TIMER SUPPORT
#NCLVLIE	EQU	6	END ADDRESS FOR LEVEL 1 TIMER SUPPORT
#NCRGMAP	EQU	7	START ADDRESS FOR REGISTER/PSW DUMP MAP
#NCLINE1	EQU	8	COMMUNICATIONS TCB ADDRESS
#NCLNDR	EQU	9	UTILITY LOADER TCB ADDRESS
#NCUPUTL	EQU	10	UTILITY ATTENTION TCB ADDRESS
*		11	SPARE
#NCLCB	EQU	12	COMMUNICATIONS LCB POINTER
#NCCOMTR	EQU	13	COMMUNICATIONS TRACE POINTER
#NCLVL	EQU	14	NUCLEUS LEVEL
#NCLVL4	EQU	15	HAZEL4/AP INTERFACE TABLES

IBM Internal Use Only

#NCPF	EQU	16	HAZEL4/AP PF TABLE
#NCEDXDB	EQU	17	HAZEL4/NATDB EDX DEBUG CONTROL BLOCK
#NCJIBDB	EQU	18	HAZEL4/NATDB JIB DEBUG CONTROL BLOCK
#NCCCB	EQU	19	CCB
#NCCNCL	EQU	20	CANCEL QCB
#NCCSCP	EQU	21	CSCP
#NCESFP	EQU	22	ESF
#NCDISKQ	EQU	23	NEW WORK DISK QUEUE
#NCDISKO	EQU	24	OLD WORK DISK QUEUE
#NCIMC	EQU	25	INTELLIGENT MACROFUNCTION CARD BASE
#NCIMCK	EQU	26	INTELLIGENT MACROFUNCTION CARD KEY
#NCENQ	EQU	27	SUPERVISOR ENQ ENTRY POINT
#NCDEQ	EQU	28	SUPERVISOR DEQ ENTRY POINT
#NCWAITO	EQU	29	INTERNAL WAIT/TIMEOUT ENTRY POINT
#NCTSKTR	EQU	30	TASK TRACE TABLE
#NCAPCMP	EQU	31	AP COMPATIBILITY TABLE

B.1.4 Nucleus Directory in Hazel/Ariel

NUCDIRC	DATA	A(\$EXEC)	50	EDX S/W STOP ADDRESS
	DATA	A(\$TRPTR)	51	CURRENT TRACE ADDRESS
	DATA	A(\$BRINGUP)	52	TRACE START ADDRESS
	DC	H'256'	53	TRACE SIZE
	DATA	A(\$READYQ)	54	READYQ POINTER
	DATA	A(\$TMICAN)	55	INTERVAL TIMER QUEUE
	DATA	A(\$TIMEND)	56	LVL1 SUPPORT ADDR HI
	DATA	A(\$REGMAP)	57	REGISTER / PSW DUMP
	DATA	A(\$LINE01)	58	COMMUNICATIONS TASK
	DATA	A(\$LNLOAD)	59	LOADER TASK
	DATA	A(\$SVUTIL)	5A	ATTENTION TASK
	DC	H'0'	5B	SPARE
	DATA	A(LCB)	5C	LCB POINTER
COMPTPR	DC	X'A000'	5D	COMM TRACE POINTER
\$NUCLVL	DATA	X'0003' NOV 29/86	5E	NUCLEUS LEVEL 3
	DC	AL2(\$APTB)	5F	HAZEL4/AP I/F TABLES
	DC	AL2(\$PFTB)	60	HAZEL4/AP PF TABLES
	DATA	A(\$EDFLGS)	61	EDX DEBUG CONTROL BLK
	DATA	A(\$JDFLGS)	62	JIB DEBUG CONTROL BLK
	DATA	A(\$SYSLOG1)	63	CCB 1
	DATA	A(\$CNCLQCB)	64	CANCEL QCB
\$CSCPTR	DC	H'0'	65	CSCP ADDRESS
\$ESFPTR	DC	H'0'	66	ESFPTRA ADDRESS
	DC	H'0'	67	NOT USED
	DC	H'0'	68	NOT USED
	DC	H'0'	69	NOT USED
	DC	H'0'	6A	NOT USED
	DATA	A(\$ENQ)	6B	SUPERVISOR \$ENQ
	DATA	A(\$DEQ)	6C	SUPERVISOR \$DEQ
	DATA	A(\$WAITO)	6D	INTERNAL WAIT ENTRY
	DATA	A(\$CSANYERR)	6E	16 C/S ANY ERROR LOG
	DATA	A(\$APDIR)	6F	AP DIRECTORY ADDRESS
	DC	AL2(\$APDISTB)	70	HAZEL4/DIS-AP IF TBL
	DATA	A(\$DISBIS)	71	BLOCK INTR STRUCTURE
	DATA	A(\$IDBPTR)	72	DIS INTERRUPT VECTOR

Figure 59. Nucleus Directory in Hazel/Ariel

#NCEXEC	EQU	0	START OF INTERPRETER MAINLINE (TESTER)
#NCDBTP	EQU	1	DEBUG CURRENT EDX TRACE POINTER
#NCDBTA	EQU	2	DEBUG EDX TRACE AREA
#NCDBTS	EQU	3	DEBUG EDX TRACE AREA SIZE
#NCRDYQ	EQU	4	SUPERVISOR READY Q
#NCLVL1	EQU	5	START ADDRESS FOR LEVEL 1 TIMER SUPPORT
#NCLVLIE	EQU	6	END ADDRESS FOR LEVEL 1 TIMER SUPPORT
#NCRGMAP	EQU	7	START ADDRESS FOR REGISTER/PSW DUMP MAP
#NCLINE1	EQU	8	COMMUNICATIONS TCB ADDRESS
#NCLNLDL	EQU	9	UTILITY LOADER TCB ADDRESS

IBM Internal Use Only

#NCUPUTL	EQU	10	UTILITY ATTENTION TCB ADDRESS
*		11	SPARE
#NCLCB	EQU	12	COMMUNICATIONS LCB POINTER
#NCCOMTR	EQU	13	COMMUNICATIONS TRACE POINTER
#NCLVL	EQU	14	NUCLEUS LEVEL
#NCLVL4	EQU	15	HAZEL4/AP INTERFACE TABLES
#NCPF	EQU	16	HAZEL4/AP PF TABLE
#NCEDXDB	EQU	17	HAZEL4/NAIDB EDX DEBUG CONTROL BLOCK
#NCJIBDB	EQU	18	HAZEL4/NAIDB JIB DEBUG CONTROL BLOCK
#NCCCB	EQU	19	CCB
#NCCNCL	EQU	20	CANCEL QCB
#NCCSCP	EQU	21	CSCP
#NCESFP	EQU	22	ESF
*	EQU	23	NOT USED
*	EQU	24	NOT USED
*	EQU	25	NOT USED
*	EQU	26	NOT USED
#HCENQ	EQU	27	SUPERVISOR ENQ ENTRY POINT
#NCDEQ	EQU	28	SUPERVISOR DEQ ENTRY POINT
#HCWAITO	EQU	29	INTERNAL WAIT/TIMEOUT ENTRY POINT
#NCCSERR	EQU	30	16 C/S ANY ERROR LOG
#HCAPDIR	EQU	31	AP TRIGGER WORD ADDRESS DIRECTORY
#NCLVL3	EQU	32	HAZEL4/APDIS INTERFACE TABLES
#HCBIS	EQU	33	DIS BLOCK INTERRUPT STRUCTURE
#NCVECTB	EQU	34	DIS INTERRUPT VECTOR TABLE

B.2 OP CODE INSTRUCTION TABLE

00	#NOP	EQU	0	
01-11	INVALID		1-17	
12	#GCMD	EQU	18	GENERAL COMMAND

B.2.1 System Commands

13	#WAITOT	EQU	19	WAIT WITH TIMEOUT
14	#WAITOTR	EQU	20	RESET/WAIT WITH TIMEOUT
15	#SATTACH	EQU	21	ATTACH TASK
16	#SDETACH	EQU	22	DETACH TASK
17	#SWAIT	EQU	23	WAIT FOR EVENT
18	#SWAITR	EQU	24	RESET EVENT
19	#SPOST	EQU	25	POST EVENT
1A	#SRESETE	EQU	26	RESET EVENT
1B	#SENQ	EQU	27	ENQUEUE RESOURCE
1C	#SDEQ	EQU	28	DEQUEUE RESOURCE
1D	#ENDATTN	EQU	29	END ATTENTION TASK
1E	#GETMAIN	EQU	30	GETMAIN
1F	#FREMAIN	EQU	31	FREMAIN
20	#DISKRW	EQU	32	DISK READ/WRITE
21	#LOADH	EQU	33	HOST/DISK LOAD
22	#PROGSTOP	EQU	34	PROGRAM STOP
23-24	INVALID		35-36	

IBM Internal Use Only

B.2.2 Terminal Input/Output Control

25	#ENQDEQT	EQU	37	ENQ/DEQ A TERMINAL
26	#PRNTEXT	EQU	38	PRINTTEXT
27	#READTXL	EQU	39	READ TEXT LINE
28	#PRNTNUM	EQU	40	PRINT NUMERIC DATA (WORD)
29	#PRNTNU4	EQU	41	PRINT NUMERIC DATA (DBLWORD)
2A	#SKLNSP	EQU	42	SKIPS AND SPACES
2B	#PRTIME	EQU	43	PRINT TIME/DAY/DATE
2C	#GTVALUE	EQU	44	READ NUMERIC DATA (WORD)
2D	#GTVALU4	EQU	45	READ NUMERIC DATA (DBLWORD)
2E	#QUESTN	EQU	46	QUESTION
2F	#READTXT	EQU	47	READ TEXT WORD
30-31	invalid		48-49	

B.2.3 Arithmetic Operations

PRECISION = 222 (opnd1=2 bytes, opnd2=2 bytes, result=2 bytes)

32	#A22	EQU	50	ADD
33	#A222	EQU	51	
34	#A222C	EQU	52	
35	#S22	EQU	53	SUBTRACT
36	#S222	EQU	54	
37	#S222C	EQU	55	
38	#M222	EQU	56	MULTIPLY
39	#M222C	EQU	57	
3A	#D222	EQU	58	DIVIDE
3B	#D222C	EQU	59	

PRECISION = 424 (opnd1=4 bytes, opnd2=2 bytes, result=4 bytes)

3C	#A424	EQU	60	ADD
3D	#A424C	EQU	61	
3E	#S424	EQU	62	SUBTRACT
3F	#S424C	EQU	63	
40	#M424	EQU	64	MULTIPLY
41	#M424C	EQU	65	
42	#D424	EQU	66	DIVIDE
43	#D424C	EQU	67	

PRECISION = 444 (opnd1=4 bytes, opnd2=4 bytes, result=4 bytes)

44	#A444	EQU	68	ADD
45	#A444C	EQU	69	
46	#S444	EQU	70	SUBTRACT
47	#S444C	EQU	71	
48	#M444	EQU	72	MULTIPLY
49	#M444C	EQU	73	
4A	#D444	EQU	74	DIVIDE
4B	#D444C	EQU	75	

PRECISION = 224 (opnd1=2 bytes, opnd2=2 bytes, result=4 bytes)

4C	#A224R	EQU	76	ADD
4D	#A224CR	EQU	77	
4E	#S224R	EQU	78	SUBTRACT
4F	#S224CR	EQU	79	
50	#M224R	EQU	80	MULTIPLY
51	#M224CR	EQU	81	
52	#D224R	EQU	82	DIVIDE
53	#D224CR	EQU	83	

IBM Internal Use Only

PRECISION = 422 (opnd1=4 bytes, opnd2=2 bytes, result=2 bytes)

54	#D422R	FQU	84	
55	#D422CR	EQU	85	DIVIDE

B.2.4 Vector Addition

56	#AV222CR	EQU	86	
57	#AV424CR	EQU	87	
58	#AV444CR	EQU	88	
59	#AV224CR	EQU	89	ADD VECTOR

B.2.5 Logical Operations

5A-5B	INVALID		90-91	
5C	#MOV2	EQU	92	
5D	#MOV2C	EQU	93	
5E	#MOV4	EQU	94	
5F	#MOV4C	EQU	95	
60	#TIME	EQU	96	
61	#MAPAD	EQU	97	
62	#INCANCL	EQU	98	
63	INVALID	EQU	99	
64	#AND2XX	EQU	100	
65	#AND2XX	EQU	101	
66	#AND2	EQU	102	
67	#AND2XX	EQU	103	
68	#AND4XX	EQU	104	
69	#AND4XX	EQU	105	
6A	#AND4	EQU	106	
6B	#AND4XX	EQU	107	
6C-6F	INVALID	EQU	108-111	
70	#IOR2XX	EQU	112	
71	#IOR2XX	EQU	113	
72	#IOR2	EQU	114	
73	#IOR2XX	EQU	115	
74	#IOR4XX	EQU	116	
75	#IOR4XX	EQU	117	
76	#IOR4	EQU	118	
77	#IOR4XX	EQU	119	
78-7B	INVALID		120-123	
7C	#EOR2XX	EQU	124	
7D	#EOR2XX	EQU	125	
7E	#EOR2	EQU	126	
7F	#EOR2XX	EQU	127	
80	#EOR4XX	EQU	128	
81	#EOR4XX	EQU	129	
82	#EOR4	EQU	130	
83	#EOR4XX	EQU	131	
84-87	INVALID		132-135	
88	#SHR2XX	EQU	136	
89	#SHR2XX	EQU	137	
8A	#SHR2	EQU	138	
8B	#SHR2XX	EQU	139	
8C	#SHR4XX	EQU	140	
8D	#SHR4XX	EQU	141	
8E	#SHR4	EQU	142	
8F	#SHR4XX	EQU	143	
90-93	INVALID		144-147	
94	#SHL2XX	EQU	148	
95	#SHL2XX	EQU	149	
96	#SHL2	EQU	150	
97	#SHL2XX	EQU	151	
98	#SHL4XX	EQU	152	
99	#SHL4XX	EQU	153	

IBM Internal Use Only

9A	#SHL4	EQU	154
9B	#SHL4XX	EQU	155

B.2.6 Loop Operations

9C	#DOLOOP	EQU	156	DO LOOP
9D	#CONTINU	EQU	157	DO LOOP END

B.2.7 Subroutine Organization

9E	#CALL	EQU	158	CALL SUBROUTINE
9F	#RETURN	EQU	159	RETURN FROM SUBROUTINE

B.2.8 Branch Instructions

A0	#BRANCH	EQU	160	UNCONDITIONAL BRANCH
A1	#CGOTO	EQU	161	COMPUTED GOTO
A2	#IFW	EQU	162	BRANCH ON CONDITION (WORD)
A3	INVALID		163	
A4	#IFDW	EQU	164	BRANCH ON CONDITION (DBL WORD)
A5	#COMPE	EQU	165	COMPARE - BRANCH IF EQUAL
A6	#COMPNE	EQU	166	COMPARE - BRANCH IF NOT EQUAL
A7-A8	INVALID		167-168	

B.2.9 Timer Instructions

A9	#STIMER	EQU	169	START SOFTWARE TASK TIMER
AA	#WAITIME	EQU	170	WAIT ON SOFTWARE TIMER
AB	#INTIME	EQU	171	LOG TIME INTERVAL DATA
AC	#INTIMEX	EQU	172	LOG TIME INTERVAL DATA, INDEXED
AD	#GETIME	EQU	173	GET TIME AND DATE

B.2.10 User Exit Instruction

AE	#USER	EQU	174	EXECUTE USER ROUTINE
AF-B0	INVALID		175-176	

B.2.11 Conversion Commands

B1	#EBCVT	EQU	177	FORMAT CONVERSION
----	--------	-----	-----	-------------------

B.2.12 Distributed Interface Channel I/O

B2	#DISWRIT	EQU	178	DIS WRITE
B3	#DISREAD	EQU	179	DIS READ
B4	#DISIO	EQU	180	DISIO
B5-BA	INVALID		181-186	

IBM Internal Use Only

B.2.13 APU Trig Support

BA	#TRIG	EQU	186	TRIG, SQRT, EXP, LN, PWR
BB-C0	INVALID		187-191	

B.2.14 ESF ODB Access Commands

C0	#GETLVID	EQU	192	GETLIST
C1	INVALID		193	
C2	#ODPGVID	EQU	194	GETITEM
C3	INVALID	EQU	195	
C4	#ODPPVID	EQU	196	GETITEM
C5	#ESFUCMD	EQU	197	ESF USER COMMANDS
C6-CF	INVALID		198-207	

B.2.15 ESF Stack Commands

D0	#TUCK	EQU	208	PUSH STACK
D1	#POP	EQU	209	POP STACK
D2	#FLOATXT	EQU	210	FLOAT -> TEXT

B.2.16 TP Communication

D3	#SEND	EQU	211	COMMUNICATION SEND
D4	#RECEIVE	EQU	212	COMMUNICATION RECEIVE
D5	#OPEN	EQU	213	COMMUNICATION OPEN
D6	#CLOSE	EQU	214	COMMUNICATION CLOSE
D7	#MOVEXP	EQU	215	CROSS PARTITION MOVE
D8-DC	INVALID		216-218	

B.2.17 Floating Point Arithmetic Commands

DB	#IFDW	EQU	219	FLOATING POINT IF
DC	INVALID		219	
DD	#MFP4	EQU	221	FLOATING POINT MOVE (ESF)
DE	#MFP4C	EQU	222	FLOATING POINT MOVE (ESF)
DF	#FPCONV	EQU	223	FLOATING POINT CONVERSION
E0	#FA222	EQU	224	FLOATING POINT ADD
E1-E7	INVALID		225-231	
E8	#FS222	EQU	232	FLOATING POINT SUBTRACT
E9-EF	INVALID		233-239	
F0	#FM222	EQU	240	FLOATING POINT MULTIPLY
F1-F7	INVALID		241-247	
F8	#FD222	EQU	248	FLOATING POINT DIVIDE
F9-FD	INVALID		249-253	
FE	#CHKEY	EQU	254	EDX CHANGE KEY
FF	#EDBUG	EQU	255	EDX DEBUG OPCODE

B.3 TASK CONTROL BLOCK (TCB)

00	#TCBC0	EQU	0	TCB CODE WORD
01	#TCBC02	EQU	1	TCB CODE WORD 2
02	#TCBECB	EQU	0	ECB
03	#TCBS0	EQU	3	SAVE LOC FOR #R0, #R1
04	#TCBS2	EQU	4	#R2, #R3

(3WDS)

IBM Internal Use Only

05	#TCBS4	EQU	5	#R4, #R5	
06	#TCBS6	EQU	6	#R6, #R7	TCB ADDR
07	#TCBS8	EQU	7	#R8, #R9	
08	#TCBS10	EQU	8	#R10, #R11	EDX/J IAR
09	#TCBS12	EQU	9	#R12, #R13	
0A	#TCBS14	EQU	10	#R14, #R15	
0B	#TCBSKY	EQU	11	KEYS - CURRENT	EDX CMD
0C	#TCBPRI	EQU	12	TASK PRIORITY	
0D	#TCBCHA	EQU	13	CHAIN LOCATION	(2WDS)
0E	#TCBEC	EQU	15	TASK END ECB	(3WDS)
12	#TCBREG	EQU	18	PIR TO TASK REG BLOCK	
13	#TCB#1	EQU	19	TASK INDEX REG #1	
14	#TCB#2	EQU	20	TASK INDEX REG #2	
15	#TCB#12	EQU	21	POINTER TO TASK REGISTER #1	
16	#TC1	EQU	22	TASK WORK LOCATION 1	
17	#TC2	EQU	23	LOCATION 2	
18	#TC3	EQU	24	LOCATION 3	
19	#TC4	EQU	25	LOCATION 4	
1A	#TC5	EQU	26	LOCATION 5	
1B	#TC6	EQU	27	LOCATION 6	
1C	#TC7	EQU	28	LOCATION 7	
1D	#TC8	EQU	29	LOCATION 8	
1E	#TCBTIM	EQU	30	TIMER VALUE - USEC FLD	
1F	#TCBTIM1	EQU	31	TIMER VALUE - MSEC FLD	
20	#TCBTCH	EQU	32	TIMER CHAIN LOCATION	(2WDS)
22	#TCBTECB	EQU	34	TIMER ECB	(3WDS)
25	#TCBTOUT	EQU	37	WAIT TIMEOUT ECB ADDR	(2WDS)
27	#TCBCCB	EQU	39	CCB ADDRESS	
28	#TCBPLP	EQU	40	PROGRAM LOAD POINT	
29	#TCBFLGS	EQU	41	TASK CONTROL FLAGS	
2A	#TCBPTR	EQU	42	PTR TO PREVIOUS TCB	
2B	#TCBADS	EQU	43	TASK TARGET ADDRESS KEY	
2C	#TCBMT	EQU	44	ADDR OF WAIT CHAIN HEAD	
2D	#TCBMTKY	EQU	45	KEY OF WAIT CHAIN HEAD	
2E	#TERMERR	EQU	46	ADDR OF TERMINAL ERROR ROUTINE	
2F	#TCBLEN	EQU	47	LENGTH OF TASK CONTROL BLOCK	
00	#1	EQU	0	TASK REGISTER #1 DISPLACEMENT	
01	#2	EQU	1	TASK REGISTER #2 DISPLACEMENT	

For every equate starting with a #, there is a corresponding equate starting with a @ that is twice the one starting with a #.

B.4 CONSOLE CONTROL BLOCK (CCB)

00	#CCBCODE	EQU	0	CCB CODE	
00	#CCBECB	EQU	0	ECB	(3WDS)
03	#CCBQCB	EQU	3	QCB CODE WORD	
04	#CCBQCKY	EQU	4	.. CHAIN WORD	
05	#CCBQCBC	EQU	5	.. CHAIN WORD KEY	(5WDS)
06	#CCBQCKY	EQU	6	.. TCB OWNER	
07	#CCBQCBT	EQU	7	.. TCB OWNER KEY	
08	#CCBQUIN	EQU	8	TASK IN CONTROL OF CCB	
09	#CCBBOTM	EQU	9	NHIST / BOTTOM MARGINS	
0A	#CCBFLGS	EQU	10	LOGICAL LEFT MARGIN / LINE LENGTH	
0B	#CCBIICB	EQU	11	MASTER COPY OF ABOVE 8 PARMS	(2WDS)
0D	#CCBPMAX	EQU	13	MAXIMUM PAGE SIZE / MAXIMUM LINE SIZE	(4WDS)
0E	#CCBNAME	EQU	14	CCB EBCDIC NAME FIELD	
12	#CCBOPTS	EQU	18	EDX COMMAND OPTION FOR TERMINAL USE	
13	#CCBERCD	EQU	19	ERROR CODE / ERROR COUNT	
14	#CCBRTNC	EQU	20	RETURN CODE / WORK1 FIELD	
15	#CCBWRK2	EQU	21	WORK2 / WORK3 FIELDS	
16	#CCBMBDV	EQU	22	ADDRESS OF VIDEO MDB	
17	#CCBMDBR	EQU	23	ADDRESS OF RAM MDB	
18	#CCBFID	EQU	24	12 BYTE CONVERSION FIELD	(6WDS)
1E	#CCBS0	EQU	30	GENERAL REGISTER SAVE AREA	
1F	#CCBS1	EQU	31	R2, R3	
20	#CCBS2	EQU	32	R4, R5	
21	#CCBS3	EQU	33	R6, R7	
22	#CCBS4	EQU	34	R8, R9	
23	#CCBS5	EQU	35	R10, R11	
24	#CCBS6	EQU	36	R12, R13	

IBM Internal Use Only

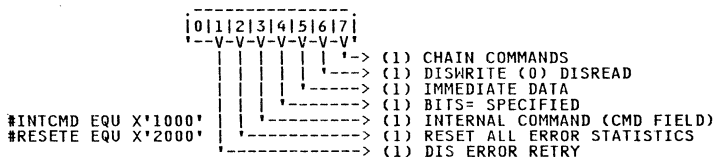
```

25 #CCBS7 EQU 37 R14,R15
26 #CCBS0 EQU 38 FIRST LEVEL 5 REGISTER SAVE AREA 5WDS
2B #CCBS1 EQU 43 SECOND LEVEL 5 REGISTER SAVE AREA 5WDS
30 #CCBS2 EQU 48 THIRD LEVEL 5 REGISTER SAVE AREA 5WDS
35 #CCBS3 EQU 53 FOURTH LEVEL 5 REGISTER SAVE AREA 5WDS
3A #CCBKYN EQU 58 KEY-IN CHARACTER / KEYBOARD INTERLOCK
3B #CCBNXTL EQU 59 NEXT LINE VALUE / CURRENT LINE VALUE
3C #CCBVCTL EQU 60 CURRENT ROLL/PICTURE CONTROL
3D #CCBRAML EQU 61 CURRENT LINE'S RAM ADDRESS
3E #CCBRAMB EQU 62 NEXT BLOCK'S RAM ADDRESS
3F #CCBDRAM EQU 63 DISPLAY PARMS LIST -RAM ADDRESS
40 #CCBDPTR EQU 64 -DATA POINTER PARMS
41 #CCBDLEN EQU 65 -LENGTH PARAMETER
42 #CCBDFLG EQU 66 -FLAGS / VIDEO CTL
43 #CCBBORG EQU 67 ADDRESS OF MESSAGE ORIGIN
44 #CCBMSIZ EQU 68 I/O MESSAGE SIZE / TBUF FLAGS
45 #CCBTPTR EQU 69 ADDRESS OF NEXT CHAR/VALUE IN MESSAGE
46 #CCBTEND EQU 70 ADDRESS OF LOGICAL END OF TBUF
47 #CCBTBUF EQU 71 64 BYTE CONSOLE I/O BUFFER

```

For every equate starting with a #, there is a corresponding equate starting with a @ that is twice the one starting with a #.

B.5 MACROFUNCTION DATA CONTROL BLOCK (MDCB)

[illegible]

		COMMAND BYTE										
		0	1	1	2	3	4	5	6	7		
			-	-	-	-	-	-	-	-		
#NOP	EQU X'0000'	0	0	0	0	0	0	0	0	0	-> NOP ... NOT INTERNAL COMMAND	
#STARTP	EQU X'0001'	0	0	0	0	0	0	0	1	-> START POLLER		
#STOPP	EQU X'0002'	0	0	0	0	0	0	1	0	-> STOP POLLER		
#READP	EQU X'0004'	0	0	0	0	1	0	0	0	-> READ POLL LIST		
#UPDTP	EQU X'0008'	0	0	0	1	0	0	0	0	-> UPDATE POLL LIST		
#RADE	EQU X'0010'	0	0	0	0	0	0	0	0	-> READ ERRORS		
#CARD	EQU X'0020'	0	0	1	0	0	0	0	0	-> CARD LIST		
#FCARD	EQU X'0040'	0	1	0	0	0	0	0	0	-> FIND CARD		
#EXRAM	EQU X'0080'	1	0	0	0	0	0	0	0	-> EXECUTE RAM CODE AT X'2000'		

IBM Internal Use Only

For every equate starting with a #, there is a corresponding equate starting with a @ that is twice the one starting with a #.

B.6 MACROFUNCTION DATA BLOCK (MDB)

00	#MDBIDAT	EQU	0	DATA FIELD / DATA FIELD POINTER
01	#MDBUV	EQU	1	U/V BITS
02	#MDBCTC	EQU	2	RCTC / CTC
03	#MDBLEN	EQU	3	R/W LENGTH (WORDS)
04	#MDBLSA	EQU	4	LSA / FLAGS
05	#MDBTECB	EQU	5	SAVE TCB ADDRESS
06	#MDBRTNC	EQU	6	RETURN CODE
07	#MDBSAVL	EQU	7	GENERAL SAVE AREA
08	#MDBMUV	EQU	8	MASTER U/V BITS
09	#MDBTYPE	EQU	9	MACROFUNCTION TYPE / RESERVED
0A	#MDBSIZE	EQU	10	LENGTH OF MDB TABLE

For every equate starting with a #, there is a corresponding equate starting with a @ that is twice the one starting with a #.

B.7 INTERRUPT DATA BLOCK (IDB)

00	#IDBECB	EQU	0	EVENT CONTROL BLOCK (3WDS)
03	#IDBADR	EQU	3	INTERRUPT BIT ADDRESS (BLOCK/BIT)
04	#IDBNAM	EQU	4	INTERRUPT BIT NAME (NUMERIC PART ONLY)
05	#IDBGECB	EQU	5	GENERAL ECB ADDRESS
06	#IDBGKEY	EQU	6	GENERAL ECB KEY
07	#IDBSIZE	EQU	7	NUMBER OF WORDS PER IDB ENTRY.

For every equate starting with a #, there is a corresponding equate starting with a @ that is twice the one starting with a #.

B.8 PROGRAM HEADER

00	#PRGORG	EQU	0	PROGRAM ORIGIN
01	#PRGNAM	EQU	1	PROGRAM NAME (4WDS)
05	#PRGLEN	EQU	5	PROGRAM LENGTH (IN WORDS)
06	#PRGTCB	EQU	6	ADDR OF MAIN TASK CONTROL BLOCK
07	#PRGSFL	EQU	7	ADDR OF LOCAL ATTNLIST COMMAND
08	#PRGEECB	EQU	8	ADDR OF PROGRAM END ECB
09	#PRGCEKEY	EQU	9	PROGRAM END ECB KEY
0A	#PRGCCB	EQU	10	ADDR OF CONSOLE CONTROL BLOCK
0B	#PRGMDBC	EQU	11	ADDR OF MDB COUNT / CHAIN
0C	#PRGRES	EQU	12	1 WORD RESERVED
0D	#PRGINTC	EQU	13	ADDR OF IDB COUNT / CHAIN
0E	#PRGHEAD	EQU	14	SIZE OF PROGRAM HEADER
0F	#PRGSCOM	EQU	15	ADDRESS OF SYSCOM
10	#PRGVERF	EQU	16	PROGRAM <> DATA VERIFICATION
11	#PRGLVL	EQU	17	PROGRAM LEVEL <> NUCLEUS VERIFICATION
12	#PRGCLOS	EQU	18	COMMUNICATION OPEN / CLOSE INDICATOR
13	#PRGHEND	EQU	19	LENGTH OF PARAMETER LIST

For every equate starting with a #, there is a corresponding equate starting with a @ that is twice the one starting with a #.

IBM Internal Use Only

APPENDIX C. CHARACTER SETS

C.1 EBCDIC CHARACTER SET

		Low Order Nibble															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	0	NUL	SOH	STX	ETX	PF	HT	LC	DEL			SMM	VT	FF	CR	SO	SI
H	1	DLE	DC1	DC2	TM	RES	NL	BS	IL	CAN	EM	CC	CU1	IFS	IGS	IRS	IUS
i	2	DS	SOS	FS		BYP	LF	ETB	ESC			SM	CU2		ENQ	ACK	BEL
g	3			SYN		PN	RS	UC	EOT				CU3	DC4	NAK		SUB
h	4	Sp										¢	.	<	(	+	
	5	&										!	\$	*	)	;	~
	6	-	/										,	%	_	>	?
N	7											:	#	@	'	=	"
i	8		a	b	c	d	e	f	g	h	i						
b	9		j	k	l	m	n	o	p	q	r						
b	A			s	t	u	v	w	x	y	z		L	l	[	≥	•
l	B	o	1	2	3	4	5	6	7	8	9		J	j	]	#	_
e	C		A	B	C	D	E	F	G	H	I			T			
	D		J	K	L	M	N	O	P	Q	R						
	E	\		S	T	U	V	W	X	Y	Z			†			
	F	0	1	2	3	4	5	6	7	8	9						

IBM Internal Use Only

C.2 ASCII CHARACTER SET

		Low Order Nibble															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
H	1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
i	2	Sp	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
g	3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
h	4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
	6	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
	7	p	q	r	s	t	u	v	w	x	y	z	{				DEL

Note: The most significant bit of the ASCII code is used as a parity bit.

IBM Internal Use Only

APPENDIX D. DISK DIRECTORY LAYOUT

DCE: D I R E C T O R Y C O N T R O L E N T R Y															
---	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

0	1	2	3	4	5	6	7	8	----->15						
#LIB REC		#FREE REC		#DME	#FSE	#REC IN DIR	UNUSED DIR BYTES								
FPDS		FPDE		FPDM	FPDF	PFDD	FPDB								

DME: D I R E C T O R Y M E M B E R E N T R Y															
---	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14/15
DATASET NAME (8 characters)				REL STRT REC#	REL END REC#	MEM TYPE	PROG LOAD POINT	MEM BYTE SIZE	--	PROG ENTRY POINT	RLD WORD SIZE	-/-/-		
FPMN				FPMS	FPME	FPMT	FPMA	FPMB		FPMP	FPMR			

FSE: F R E E S P A C E E N T R Y															
---	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

0	1	2	3
FIRST REC FSE	LAST REC FSE		
FSEFFS	FSELF5		

IBM Internal Use Only

DISK FORMATS

D I S K F O R M A T S				
	TYPE 1	TYPE 2	TYPE 2D	HARD FILE
BYTES/SECTOR	128 . 256	128 . 256	128 . 256	256
SECTORS/HEAD	26 . 15	26 . 15	52 . 30	32
LRECS/HEAD	13 . 15	13 . 15	26 . 30	32
BYTES/HEAD	3328 . 3840	3328 . 3840	6656 . 7680	8192
HEADS/CYL	1	2	2	4
LRECS/CYL	13 . 15	26 . 30	52 . 60	128
BYTES/CYL	3328 . 3840	6656 . 7680	13,312 / 15,360	32768
CYLS/VOLUME	74	74	74	303
LRECS/VOLUME	962 . 1110	1924 . 2220	3848 . 4440	38784
BYTES/VOLUME	246,272 / 284,160	492,544 / 568,320	985,088 / 1,136,640	9,928,704

CTS = CYLINDER / TRACK / SECTOR

0	C	C	C	H	H	S	S
---	---	---	---	---	---	---	---

```

'--V---'--V---V---'
|           |           |
|           |           |-----> SECTOR# .....(1->32)
|           |           |-----> HEAD# .....(0->3)
|           |           |-----> CYLINDER# ... (0->303)

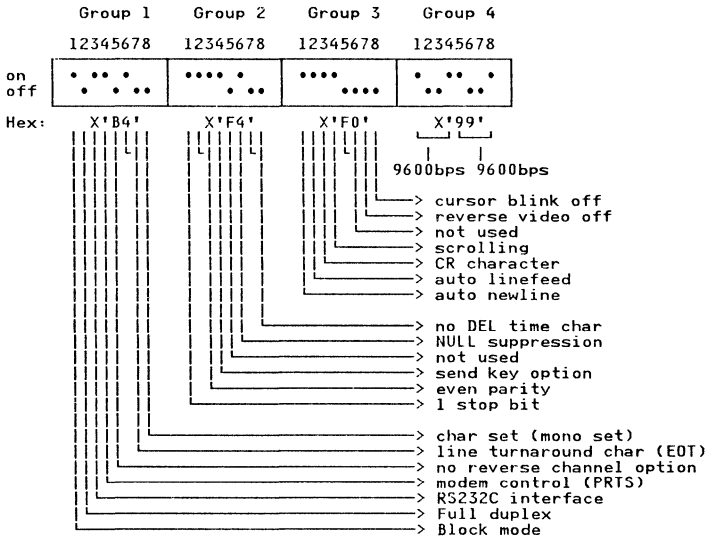
```

IBM Internal Use Only

APPENDIX E. 3101 OPERATION

The following description is intended to be used as an guide on the 3101 terminal setup for with the DSS IV system.

This terminal occupies the third port on the CSS (Communication Subsystem) card supported by the AP (Attached Processor). and is supported only in BLOCK mode. To select this and other 3101 features, an array of setup switches located on the keyboard is used. For the 3101 to operate correctly within the system, the setup switches must be in the following position.



For further information of the 3101 terminal, the user should refer to the 'IBM 3101 Display Terminal Description' (Form Number GA18-2033).

IBM Internal Use Only

IBM Internal Use Only

APPENDIX F. HAZEL/4 - HAZEL/ARIEL DIS SUPPORT COMPATIBILITY

The EDX DIS support for Hazel/Ariel is different from the DIS support for Hazel/4. The following are the changes made to DIS support in the EDX language:

- the syntax of the following existing EDX commands was changed: DIS-READ, DISWRITE and LSA
- new commands were added: DISIO and MDCB
- the function of some commands was changed: DISREAD and DISWRITE
- internal data structures were changed (MDB was replaced by MDCB)

Because of the internal data structure changes, assembled programs which ran on Hazel/4 will not run on Hazel/Ariel without reassembly.

IBM Internal Use Only

F.1 EDX SUPPORT FOR DIS

EDX supports the DIS channel with the following statements and commands:

- LSA** Defines an MDB or MDCB data structure (depending if assembled for Hazel/4 or Hazel/Ariel). The MDB or MDCB data structures contain the data transfer parameters. The LSA statement defines only some of the MDCB or MDB parameters and the DISREAD or DISWRITE commands specify the rest. The LSA statement must be used with the DISREAD and DISWRITE commands.
- INTBIT** Defines a structure which the operating system needs to associate an interrupt from a specific point on a I/O card to an EDX task(s).
- DISWRITE** EDX command to write one or more bits, bytes or words to a macrofunction card.
- DISREAD** EDX command to read one or more bits, bytes or words from a macrofunction card.
- DISIO** A new EDX command which supports the same functions as DISREAD and DISWRITE, but in addition it allows the DIS I/O requests to be chained for a more efficient use of the channel interface. The chaining feature allows to execute a theoretically infinite number of DIS I/O requests by issuing a single DISIO command. Another new feature of DISIO is that the user is given an option to overlap the DIS I/O processing and EDX execution. If this option is used, the EDX instructions following a DIS I/O instruction will be executed immediately, not after the actual data transfer is completed.
- MDCB** A new EDX statement which creates the MDCB data structure. The MDCB defines all the data transfer parameters. In contrast, the LSA statement defines only some of the MDCB parameters and the DISREAD or DISWRITE commands specify the rest. The MDCB statement must be used with the DISIO command.

The following data structures are used by the operating system to execute DIS I/O instructions:

- MDB** Is a data structure which the operating system needs for handling and directing I/O instructions (DISWRITE and DISREAD) to a specific macrofunction card. See "MDB Description / Layout" on page 266.
- MDCB** Is a new data structure similar to the MDB structure. The MDCB data structure was created to allow chaining of the DIS I/O requests. Unlike in the MDB structure, all parameters in the MDCB structure are unpacked; that is each parameter occupies a single word. See "MDCB Description / Layout" on page 267.
- IDB** Is a data structure which the operating system needs to associate an interrupt from a specific point on a I/O card to an EDX task(s). See "IDB Description / Layout" on page 269.

The DISIO command and the MDCB statement have been developed to compensate for the slower data transfer rate in the Hazel/Ariel serial DIS channel. In addition to this, an attempt was made to simplify the application code by changing the syntax of the EDX instructions and making it easier to modify the MDCB (formerly MDB).

IBM Internal Use Only

F.2 COMPATIBILITY

The discussion on compatibility of the Hazel/4 and Hazel/Ariel EDX DIS support will be conducted under three topics: syntax, function and performance.

F.2.1 Syntax Compatibility

The EDX syntax of the DISREAD, DISWRITE, LSA and INTBIT has not changed to maintain source code compatibility. However, some new keywords were added to the LSA, DISREAD and DISWRITE commands.

F.2.1.1 LSA

The LSA statement has a new optional keyword, 'LSA=', which was added to alleviate the awkwardness of defining an I/O address in hexadecimal. Future programs can now specify the LSA address in decimal.

The Master U/V bits parameter on the LSA statement is now ignored.

F.2.1.2 DISREAD and DISWRITE

A new EVENT= parameter was added. When it is specified, the task executing the DISIO command does not wait for the command to complete (except when COUNT=1), but continues to execute while the AP/DIS controller is executing the I/O command.

Note, that if EVENT= is specified there is a possibility that the same DIS I/O command may be re-executed before it completes. To avoid this, the task is responsible to test the I/O completion code (placed in the first word of ICB), or to wait for the ECB to be posted with the I/O completion code, before it re-executes that same I/O instruction.

If EVENT= is not specified, the task executing the DISIO command will be suspended until the DISIO command is completed.

F.2.2 Functional Compatibility

Some of the internal changes to the DIS commands will affect the operation of a small number of programs which used the DIS channel. Two areas where operational compatibility was not possible are described below.

F.2.2.1 Internal Wait

The DISWRITE, DISREAD and DISIO commands all do internal waits while the AP/DIS controller executes the I/O operation. During this time other tasks (including tasks with lower priority) are permitted to execute. Once the I/O operation is complete the AP interrupts Hazel, and the operating system restores the task. In contrast, the Hazel/4 DIS support executed the I/O operation with no breaks: no other tasks were permitted to run during the duration of a DIS I/O command.

Hazel/Ariel DIS support (unless EVENT= is specified) after executing a DIS I/O command puts the task which issued the command in the wait queue and performs a task switch if there is a task in the ready queue waiting for execution. The task which issued the DIS I/O command remains in the wait queue until the AP/DIS controller completes the data transfer. The overall affect on programs which used the DIS channel (unless EVENT= is specified)

IBM Internal Use Only

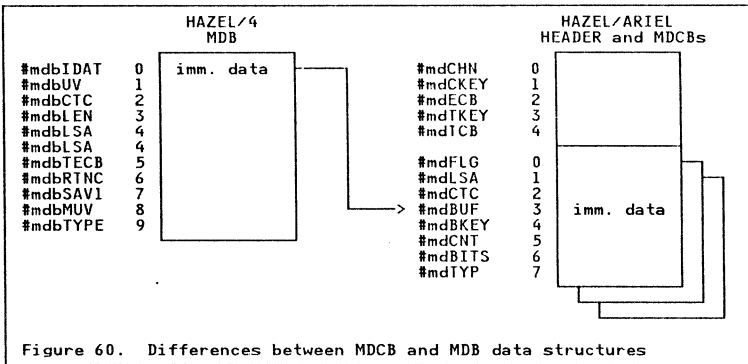
is that the task will loose control during the I/O operation and the operation will take longer.

F.2.2.2 Program Reference to MDB Fields

The MDB structure has been replaced by the MDCB structure.

The EDX programs which modify fields in the MDB in order to change the DIS data transfer parameters (e.g. LSA, CTC, bits format or count) dynamically, will need minor changes to accommodate the new MDB now called a MDCB.

The figure below shows the differences between the two structures. A particular attention is drawn to the relocation of the immediate data buffer and to the presence of the MDCB header in front of the MDCB body.



The application programs can still access the internal fields within the MDCB structure. It is recommended that, for this purpose, the P= parameter on the MDCB statement be used instead of the equates. See "MDCB - ARIEL" on page 112.

It should be noted that, unlike the MDB, the internal fields in the MDCB are not encoded (only one parameter per word) and therefore, application programs can do simple word moves to update the MDCB parameters.

F.2.2.3 DIS I/O with COUNT=1

In this case the operational compatibility with Hazel/4 was actually preserved in order to maximize the speed of the COUNT=1 DIS operations. This type of transfer departs from the usual way DIS I/O is handled in Hazel/Ariel. When DISREAD, DISWRITE or DISIO command is issued with COUNT parameter set to one (COUNT=1), all other operations (including tasks of higher priority) are locked out for the duration of the transfer. As a result, even if EVENT= is specified, the next sequential EDX instruction will not be executed until the DIS I/O instruction (with COUNT=1) is completed. Note, that the above is not true when MDCBs are chained.

F.3 DIS CHANNEL PERFORMANCE

F.3.1 Measurement Tools

Two EDX tasks were used for the DIS channel performance evaluation. See Figure 61 and Figure 62 on page 316. The task with the lower priority (the I/O task) performed DIS I/O operations and incremented the I/O count continuously. The task with the higher priority was activated every second to print and then resets the I/O count. While the DIS I/O performance was measured no other EDX task was active.

The tables in Figure 63 on page 317, Figure 64 on page 318 and Figure 65 on page 318 show the DIS I/O operations performed by the I/O task during a one second time interval.

```

This figure shows the new DIS I/O format - the DISIO command. In
Figure 63 on page 317, Figure 64 on page 318 and Figure 65 on page
318 label NEW is used to indicate that this type of command was used.

*****
* TASK = TASKTIME: PRINT AND RESET COUNT EVERY 1 SECOND.
*****
*SPACES 2
TASKTIME TASK      START1,254,TERMERR=ATTERR1
START1  EQU        *
        MOVE       DISCNT,0
        STIMER     1000,WAIT
        MOVE       DIFF,DISCNT
        PRINTNUM   DIFF,1
        PRINTTEXT  'a'
        GOTO       START1
        ENDTASK

*****
* TASK = TASKWRIT: DIS WRITE TASK USING DISIO.
*****
*SPACES 2
TASKWRIT TASK      START2,255,TERMERR=ATTERR2
START2  EQU        *
        DISIO      WRITE,ERROR=ER2,P1=ECBW
        ADD        DISCNT,1
        GOTO       START2
        ENDTASK
    
```

Figure 61. EDX Tasks Used for DIS I/O Frequency Comparison - 'NEW' format.

IBM Internal Use Only

This figure shows the old DIS I/O format - the DISWRITE command. In Figure 63 on page 317, Figure 64 on page 318 and Figure 65 on page 318 label OLD is used to indicate that this type of command was used.

```
*****
* TASK = TASKTIME: PRINT AND RESET COUNT EVERY 1 SECOND.
*****
*SPACES 2
TASKTIME TASK      START1,254,TERMERR=ATTERR1
START1 EQU *
      MOVE DISCNT,0
      STIMER 1000,WAIT
      MOVE DIFF,DISCNT
      PRINTNUM DIFF,1
      PRINTTEXT 'a'
      GOTO START1
      ENDTASK

*****
* TASK = TASKWRIT: DIS WRITE TASK USING DISWRITE.
*****
*SPACES 2
TASKWRIT TASK      START2,255,TERMERR=ATTERR2
START2 EQU *
      DISWRITE LSA1,BUF,CTC=4,COUNT=1,ERROR=ER2,P1=LSAW,P5=CNTH
      ADD DISCNT,1
      GOTO START2
      ENDTASK
```

Figure 62. EDX Tasks Used for DIS I/O Frequency Comparison - 'OLD' format.

F.3.2 Results

The following explains the acronyms used in the figures below:

- CMD** - Two DIS I/O commands were tested: read and write.
- CTC** - One or two bytes are transferred depending whether odd or even CTC is used.
- CNT** - Two values of count were used: one and ten.
- BITS** - For a given value of count both bit and byte transfers were evaluated.
- PAR** - Stands for Hazel/3 or Hazel/4 parallel channel DIS interface.
- SCA** - Stands for Hazel/4 serial channel DIS interface via the Source Communication Adapter (SCA).
- AP** - Stands for Hazel/Ariel AP based DIS interface.
- OLD** - Refers to the old format of the DIS commands, namely: DISREAD and DISWRITE.
- NEW** - Refers to the new format of the DIS commands, namely: DISIO.
- LOC** - Stands for 'local block'. It implies that a parallel rather than serial DIS channel is used.
- REM** - Stands for 'remote block'. It implies that a serial rather than parallel DIS channel is used.
- POL** - Implies that DIS interrupt polling function is active. Note the following:

IBM Internal Use Only

- for the parallel DIS channel (PAR) DIS interrupt processing is interrupt driven and therefore, the polling function is not required.
- the SCA based DIS channel (SCA) polls DIS interrupts all the time, even when no EDX task is waiting for a DIS interrupt.
- the AP based DIS channel (SCA) polls DIS interrupts only when polling is required, that is, an EDX task is waiting for a DIS interrupt.

CMD CTC CNT BITS				PAR LOC —	SCA REM POL	AP LOC —	AP LOC POL	AP REM —	AP REM POL	AP NEW LOC	AP NEW LOC	AP NEW REM	AP NEW REM
WR	6	1	—	6635	3090	4741	4316	3284	2685	4974	4911	3404	2866
WR	6	10	—	2435	625	2018	2002	710	683	2061	2048	715	688
WR	7	1	—	7025	3860	4875	4398	3970	3109	5150	5065	4142	3378
WR	7	10	—	3050	1050	2198	2183	1126	1061	2248	2233	1139	1072
WR	6	1	2,2,4	4145	1895	3261	3054	2025	1769	3379	3351	2071	1861
WR	6	1	2,8,4	3970	1855	3132	2916	1971	1738	3232	3206	2016	1817
WR	6	10	2,2,4	775	280	693	691	305	300	698	697	306	301
WR	6	10	2,8,4	715	270	636	634	293	289	640	639	294	289
WR	7	1	2,2,4	4270	2155	3389	3141	2600	2176	3512	3486	2673	2331
WR	7	1	2,8,4	4090	2105	3248	3053	2515	2124	3359	3327	2585	2262
WR	7	10	2,2,4	820	335	746	744	451	440	752	750	453	442
WR	7	10	2,8,4	755	325	678	677	426	416	683	682	428	418
RD	6	1	—	6415	3040	4711	4371	3263	2618	4904	4883	3364	2840
RD	6	10	—	2300	615	1935	1926	690	666	1971	1960	695	670
RD	7	1	—	6730	3765	4839	4621	3961	3107	5040	5011	4141	3360
RD	7	10	—	2760	1015	2140	2125	1108	1044	2184	2170	1119	1055
RD	6	1	2,2	4965	2665	4237	3958	3033	2489	4415	4383	3125	2670
RD	6	1	2,8	5880	2780	3668	3393	2731	2280	3797	3772	2809	2435
RD	6	10	2,2	1120	480	1612	1603	646	624	1638	1629	651	628
RD	6	10	2,8	1355	520	1525	1514	631	610	1548	1538	635	614
RD	7	1	2,2	5195	3225	4304	4117	3626	2886	4510	4494	3766	3121
RD	7	1	2,8	5650	3500	3747	3569	3210	2603	3881	3865	3326	2804
RD	7	10	2,2	1210	695	1755	1743	994	942	1784	1776	1003	951
RD	7	10	2,8	1585	785	1650	1641	959	911	1678	1667	966	920
AVERAGE FREQUENCY				3494	1705	2739	2600	1855	1565	2835	2814	1909	1658
% OF PARALLEL				100%	49%	78%	71%	53%	44%	81%	81%	55%	47%

Figure 63. Comparison of DIS I/O Frequencies between Parallel Channel, SCA and AP/DIS.

IBM Internal Use Only

CMD	CTC	CNT	BITS	PAR OLD LOC —	SCA OLD REM POL	AP OLD LOC —	AP OLD LOC POL	AP OLD REM —	AP OLD REM POL	AP NEW LOC —	AP NEW LOC POL	AP NEW REM —	AP NEW REM POL
WR	6	1	—	6635	3090	4741	4316	3284	2685	4974	4911	3404	2866
WR	7	1	—	7025	3860	4875	4398	3970	3109	5150	5065	4142	3378
WR	6	1	2,2,4	4145	1895	3261	3054	2025	1769	3379	3351	2071	1861
WR	6	1	2,8,4	3970	1855	3132	2916	1971	1738	3232	3206	2016	1817
WR	7	1	2,2,4	4270	2155	3389	3141	2600	2176	3512	3486	2673	2331
WR	7	1	2,8,4	4090	2105	3248	3053	2515	2124	3359	3327	2585	2262
RD	6	1	—	6415	3040	4711	4371	3263	2618	4904	4883	3364	2840
RD	7	1	—	6730	3765	4839	4621	3961	3107	5040	5011	4141	3360
RD	6	1	2,2	4965	2665	4237	3958	3033	2489	4415	4383	3125	2670
RD	6	1	2,8	5880	2780	3668	3393	2731	2280	3797	3772	2809	2435
RD	7	1	2,2	5195	3225	4304	4117	3626	2886	4510	4494	3766	3121
RD	7	1	2,8	5650	3500	3747	3569	3210	2603	3881	3865	3326	2804
AVERAGE FREQUENCY % OF PARALLEL				1573 100%	583 37%	1466 93%	1456 93%	695 44%	666 42%	1490 95%	1482 94%	700 45%	671 43%

Figure 64. Comparison of DIS I/O Frequencies between Parallel Channel, SCA and AP/DIS (DIS I/O with Count 1).

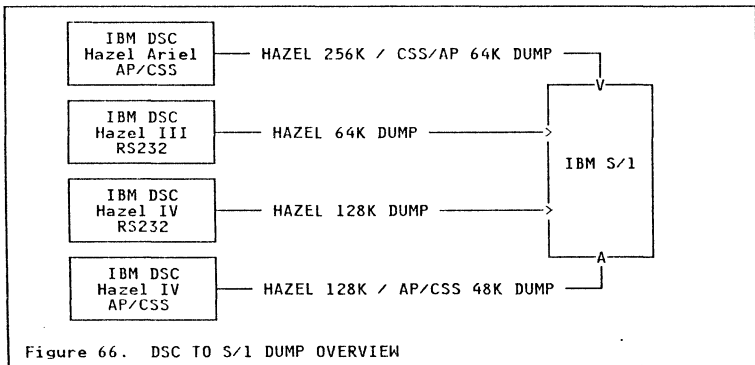
CMD	CTC	CNT	BITS	PAR OLD LOC —	SCA OLD REM POL	AP OLD LOC —	AP OLD LOC POL	AP OLD REM —	AP OLD REM POL	AP NEW LOC —	AP NEW LOC POL	AP NEW REM —	AP NEW REM POL
WR	6	10	—	2435	625	2018	2002	710	683	2061	2048	715	688
WR	7	10	—	3050	1050	2198	2183	1126	1061	2248	2233	1139	1072
WR	6	10	2,2,4	775	280	693	691	305	300	698	697	306	301
WR	6	10	2,8,4	715	270	636	634	293	289	640	639	294	289
WR	7	10	2,2,4	820	335	746	744	451	440	752	750	453	442
WR	7	10	2,8,4	755	325	678	677	426	416	683	682	428	418
RD	6	10	—	2300	615	1935	1926	690	666	1971	1960	695	670
RD	7	10	—	2760	1015	2140	2125	1108	1044	2184	2170	1119	1055
RD	6	10	2,2	1120	480	1612	1603	646	624	1638	1629	651	628
RD	6	10	2,8	1355	520	1525	1514	631	610	1548	1538	635	614
RD	7	10	2,2	1210	695	1755	1743	994	942	1784	1776	1003	951
RD	7	10	2,8	1585	785	1650	1641	959	911	1678	1667	966	920
AVERAGE FREQUENCY % OF PARALLEL				5414 100%	2828 52%	4013 74%	3515 65%	3016 56%	2419 45%	418 77%	4145 76%	3119 58%	2645 49%

Figure 65. Comparison of DIS I/O Frequencies between Parallel Channel, SCA and AP/DIS (DIS I/O with Count 10).

IBM Internal Use Only

APPENDIX G. MEMORY DUMP TO S/1 DISKETTE

This document explains how to dump the DSC memory to S/1. First it describes how to initialize diskettes for the DSC to S/1 dump. Then it shows how to start a dump from a DSC with RS232 and AP/CSS based communications. Finally, it describes how to use **DTD** (Dump to Diskette) and **UTDH** (Utility to Display or Dump to Host) S/1 utility programs to dump DSC memory to host and display the dumps on S/1 console or a local printer.



IBM Internal Use Only

G.1 DISKETTE INITIALIZATION

The S/1 utilities used for diskette initialization are: \$DASDI, \$INITDSK and \$DISKUT1. The description of the initialization consists of prompts from the S/1 utility programs and the responses entered by the user. The user responses are highlighted.

Note, that Hazel/Ariel memory dump (256K) requires 2D type diskettes.

G.1.1 \$DASDI and \$INITDSK

- > \$L \$DASDI
- ENTER DISK INITIALIZATION OPTION: 1
- ENTER DISKETTE ADDRESS IN HEX: 2 or ?¹
- INITIALIZE FOR USE WITH THE IBM EVENT DRIVEN EXECUTIVE (Y/N)? Y
- SELECT SECTOR SIZE (1=128, 2=256): 2
- FORMATTING WILL DESTROY ALL DATA ON THE DISKETTE.
- CONTINUE (Y/N)? Y or N²
- LOAD \$INITDSK (Y/N)? Y
- COMMAND (?) ID
- DEVICE ADDRESS: 2
- INITIALIZE DEVICE MAY DESTROY ALL DATA CONTINUE (Y/N)? Y or N²
- INITIALIZE DISKETTE AS MULTI-VOLUME TYPE (Y/N)? N
- NEW VOLUME LABEL: EDXDMP
- ENTER OWNER IDENTIFICATION: owner name and/or initials
- SINGLE-VOLUME TYPE DISKETTE INITIALIZED
- INITIALIZE THE VOLUME (Y/N)? Y
- MAXIMUM NUMBER OF DATA SETS: 102
- DO YOU WANT WRITE VERIFY FOR THIS VOLUME (Y/N)? Y
- ALLOCATE \$EDXNUC (Y/N)? N
- COMMAND (?) EN
- ANOTHER DISKETTE? N
- ENTER DISK INITIALIZATION OPTION: 4

G.1.2 \$DISKUT1

¹ The diskette address may not be 2 for your installation.

² To avoid erasure of data from the hard disk listen for the diskette drive to recalibrate. If you do not hear the diskette drive, enter N to abort the initialization!

IBM Internal Use Only

- > \$L \$DISKUT1
- COMMAND (?) CV EDXDMP
- COMMAND (?) AL
- MEMBER NAME: DMPDIR
- HOW MANY RECORDS? 2
- DEFAULT TYPE = DATA - OK? Y
- COMMAND (?) AL
- MEMBER NAME: DUMP
- HOW MANY RECORDS? 1028 or 2056³
- DEFAULT TYPE = DATA - OK? Y
- COMMAND (?) EN

³ Hazel/Ariel requires 2056 records Hazel/3 and Hazel/4 require only 1028 records. Note, that 2D type diskette is required to allocate 2056 records.

IBM Internal Use Only

G.2 DISABLING S/I SENDS TO DSC

A requirement for a successful DSC to S/I dump is that there be no other communication traffic between the DSC and the S/I (except for the dump). Therefore, before the dump is started, the S/I DTD program "disables" the line via which the dump is to take place. When a line is "disabled" the TP Manager in S/I prevents S/I application programs to send to the "disabled" line, messages with priority higher than 10. The priority of messages sent from the DTD program is 9.

IBM Internal Use Only

G.3 DSC MEMORY DUMP AND DISPLAY

G.3.1 STARTING DSC TO S/I DUMP

PROCEDURAL RULES

1. Only one dump per EDXDMP diskette is allowed.
2. Reuse of EDXDMP diskettes does not require reformatting.
3. To aid in the maintenance of EDXDMP diskettes the following measures are recommended:
 - Maintain a large supply (say 10) of clearly labeled EDXDMP diskettes in a box posted on a wall.
 - All the used EDXDMP diskettes should have up to date labels and be kept for future reference in a separate box.
4. It is strongly advised that a copy of this document be posted on the wall in the vicinity of the DSC's and the S/I's.

G.3.1.1 DUMP PROCEDURE FOR RS232 BASED COMMUNICATIONS

To initiate the DSC to S/I dump, perform the following steps in sequence:

1. Determine the line number of the DSC which is required by the S/I DTD program.
2. Insert EDXDMP diskette into S/I diskette drive.
3. Load the S/I DTD program and enter requested information.
4. When the DTD program prompts to start the dump on DSC go through the following steps:
 - If not already in the M/C handler, hit TEST REQ key on DSC keyboard to enter the M/C handler.
 - Hit PF8 key to arm DSC for the dump.
 - Observe the light pattern on front panel of the DSC - lights 0 to 3 should be off and lights 4 to 7 should start flashing. Flushing lights indicate that Hazel is waiting for the S/I DTD program to initiate the dump.
 - Any other light pattern on the front panel indicates a RS232 initialization error; hit the following keys: PF9 and PF8 to restart the dump.
5. Continue entering information requested by the S/I DTD program.

IBM Internal Use Only

G.4 TERMINATION / RESTART FOR RS232 BASED DUMP

When the dump ends, the dump program in DSC returns control to the M/C handler.

Hit **PF9** in case the dump abends, or to terminate the dump in progress. The **PF9** key disarms the dump and returns control to the M/C handler.

G.4.1.1 DUMP PROCEDURE FOR AP/CSS BASED COMMUNICATIONS

To initiate the DSC to S/l dump, perform the following steps in sequence:

1. Determine the line number of the DSC which is required by the S/l **DTD** program.
2. Insert **EDXDMP** diskette into S/l diskette drive.
3. Load the S/l **DTD** program and enter requested information.
4. When the **DTD** program prompts to start the dump on DSC go through the following steps:
 - Hit **PF1** key on the 3101 keyboard to enter AP Hazel Debug.
 - Enter **HA** to dump Hazel memory or **AP** to dump AP/CSS memory.
 - Enter **DU** to arm the dump. At this point AP is waiting for the S/l **DTD** program to initiate the dump.
 - To determine that the dump is armed, observe the AP light pattern - lights 0 to 3 should start flashing. Flushing lights indicate that the AP is waiting for the S/l **DTD** program to initiate the dump.
5. Continue entering information requested by the S/l **DTD** program.

G.4.1.2 TERMINATION / RESTART FOR AP/CSS BASED DUMP

When the dump ends, the dump program in AP returns control to the AP Hazel Debug.

Hit **ENTER** in case the dump abends, or to terminate the dump in progress. The **ENTER** key disarms the dump and returns control to Hazel Debug.

G.4.2 DISPLAYING/UPLOADING THE DUMP

Using the S/l **UTDH** (Utility to Display or Dump to Host) program, the DSC dump may be displayed on S/l console, printed on S/l printer or uploaded to S/370.

To use the **UTDH** program the following steps are required:

1. Insert **EDXDMP** diskette into S/l diskette drive.
2. Load **UTDH** program.
3. Enter '?' to display available commands.

IBM Internal Use Only

G.4.2.1 ALLOCATING DUMP DATA SET ON S/370

If the DSC dump is to be sent to the host, a dump data set must be allocated on S/370. The following is an example of how to allocate a sequential dump data set using ISPF 3.2:

```
SPACE UNITS      ==> BLKS
PRIMARY QUANT    ==> 105
SECONDARY QUANT  ==> 2
DIRECTORY BLOCKS ==> 0
RECORD FORMAT    ==> FB
RECORD LENGTH    ==> 256
BLOCK SIZE       ==> 2560
```

G.4.2.2 FORMATTING THE DUMP ON S/370

To format the dump data set uploaded from S/1 use the **DSCDUMP** clist. The clist accepts six optional key words: four with parameters and two without parameters.

The following is a list of keywords with parameters:

1. **DA(name)** - where **name** is the second qualifier of the dump data set. **DEFAULT = DSCDUMP.**
2. **SYSOUT(sysout)** - where **sysout** is the sysout class which defines the type of paper on which the formatted dump will be printed. **DEFAULT = A.**
3. **LOADLIB(loadlib)** - where **loadlib** is the fully qualified name of the data set containing the dump formatting load module. **DEFAULT = ts85834.PLI.LOAD.**
4. **LOADMEM(loadmem)** - where **loadmem** is the name of the the dump formatting load module. **DEFAULT = FRMTDUMP.**

The following is a list of keywords without parameters:

1. **HOLD** - when specified causes the formatted dump to be held on the output queue. The user may then browse it, purge it, or requeue it to the printer.
2. **DIAG** - when specified activates MSG, LIST, CONLIST and SYMLIST clist processing options.

IBM Internal Use Only

IBM Internal Use Only

APPENDIX H. S/1 COMMUNICATION TRACE

This document outlines the use of the S/1 TP Trace facility. First the creation process and attributes of the dump diskette are given. Then the options and use of the facility are described.

The trace facility was developed to collect data for each send and receive on a selected TP line. Trace data can be stored in memory, to a diskette or to a hard disk. Four traces may be stored on a single diskette. Each data area on a diskette is 256 records in length, holding data for up to 512 transactions.

When a diskette or a hard disk option is used an overrun may occur. An overrun occurs when trace buffer is filled faster than it is emptied to a diskette or a hard disk. As a result, some trace transactions may be lost.

The advantage of using the the diskette option is that a S/1 failure will not delete collected data. The advantage of tracing to memory is that trace overrun will not occur. And finally, the advantage of using the hard disk option is that overruns are less likely than with the diskette option.

There are two versions of the trace: one with a 32-transaction buffer - **TRACES** and one with a 128-transaction buffer - **TRACEB**. The 128-transaction buffer trace requires more memory but overruns are less likely to occur.

IBM Internal Use Only

H.1 DISKETTE INITIALIZATION

The S/1 utility used for diskette initialization are \$DASDI, \$INITDSK and \$DISKUT1. The description of the initialization consists of the prompts from the S/1 utility program and the responses entered by the user. The user responses are highlighted.

The user may choose to dump to a volume of the hard disk, rather than to an EDXDMP diskette. To use this option, follow **only** the procedure listed under the title 'Creating the DUMP/TRACE dataset' below. **DO NOT** follow the procedure under 'Initializing the diskette'. See the section titled 'Trace Overrun' to see when it may be appropriate to use the hard disk.

H.1.1 Initializing the diskette

- > \$L \$DASDI
- ENTER DISK INITIALIZATION OPTION: 1
- ENTER DISKETTE ADDRESS IN HEX: 2/?⁴
- INITIALIZE FOR USE WITH THE IBM EVENT DRIVEN EXECUTIVE? Y
- SELECT SECTOR SIZE (1=128, 2=256): 2
- FORMATTING WILL DESTROY ALL DATA ON THE DISKETTE. CONTINUE? Y/N⁵
- LOAD \$INITDSK? Y
- COMMAND (?) ID
- DEVICE ADDRESS: 2
- NEW VOLUME LABEL: EDXDMP
- ENTER OWNER IDENTIFICATION: owner name and/or initials
- INITIALIZE THE VOLUME? Y
- MAXIMUM NUMBER OF DATA SETS: 102
- DO YOU WANT WRITE VERIFY FOR THIS VOLUME? Y
- ALLOCATE \$EDXNUC? N
- COMMAND (?) EN
- ANOTHER DISKETTE? N
- ENTER DISK INITIALIZATION OPTION: 4

H.1.2 Creating the DUMP/TRACE dataset

- > \$L \$DISKUT1
- COMMAND (?) CV EDXDMP or CV followed by a valid hard disk volume name
- COMMAND (?) AL

⁴ The diskette address may not be 2 for your installation.

⁵ To avoid erasure of data from the hard disk listen for the diskette drive to recalibrate. If you do not hear the diskette drive, enter N to abort the initialization!

IBM Internal Use Only

- MEMBER NAME: DUMP
- HOW MANY RECORDS? 1028
- DEFAULT TYPE = DATA - OK? Y
- COMMAND (?) EN

IBM Internal Use Only

H.2 USING THE TRACE FACILITY

PROCEDURAL RULES

1. Four traces per EDXDMP diskette are allowed.
2. Reuse of EDXDMP diskettes does not require reformatting.
3. To aid in the maintenance of EDXDMP diskettes the following measures are recommended:
 - Maintain a large supply (say 10) of clearly labeled EDXDMP diskettes in a box posted on a wall.
 - All the used EDXDMP diskettes should have up to date labels and be kept for future reference in a separate box.
4. It is strongly advised that a copy of this document be posted on the wall in the vicinity of the DSC's and the S/I's.

H.2.1 Starting the Trace Programs

To initiate the S/I trace facility, perform the following steps in sequence:

1. Determine the line number of the DSC which is to be monitored.
2. Insert EDXDMP diskette into S/I diskette drive if you want a permanent copy of the trace.
3. Load the S/I TRACEB or TRACES program and enter the following commands:
 - ENTER DISK VOLUME TO BE USED OR PRESS ENTER TO USE EDXDMP DISKETTE: press 'enter' unless you are using the hard disk, in which case enter the name of the disk volume containing the 'DUMP' dataset.
 - > START
 - ENTER TC # : desired line to trace
 - DUMP TO DISKETTE? Y

If this is the first time the diskette is used with TRACES/TRACEB it will say, "INVALID HEADER FORMAT - OVERWRITE?". If the data on the diskette is no longer needed reply Y, otherwise reply N and type > START again.

- TRACE AREA NUMBER: number of desired trace area

If the area already contains data you have the option to have it overwritten or select another area.

- DESCRIPTION OF TRACE: your trace description

Note, you must enter a description.

- ALLOW WRAP ON THE DISKETTE? Y

If you enter N tracing on the diskette will terminate when the end of the trace area is reached. If you enter Y tracing on the diskette will continue at the start of the trace area when the end is reached.

- At this point TRACEB/TRACES will be collecting data for the selected line.

IBM Internal Use Only

H.2.2 Added options for diskette tracing

To accommodate the option to dump to diskette four options were added to the trace facility. These options are:

1. SDISK - Start dumping to diskette. This option was added to allow the user to start up the diskette dump task at any time without re-starting the in-memory trace. Prompts and responses are the same as when > START is entered.
2. EDISK - Stop dumping to diskette. This option was added to allow the user to stop the diskette dump task without stopping the in-memory trace.
3. DISPLAY - Display/print trace data. This option will be described in greater detail later in this document.

H.2.2.1 DISPLAY command

This command is used when you want to display trace data on a screen or optionally have it printed. When this option is selected all tracing (in-memory and diskette) is stopped. The descriptions of the traces in the various areas will be displayed and the user will be prompted to choose a trace area to display. If you don't want to display any of the areas enter 0 for the desired area. Once an area has been selected the oldest data in the area is displayed and the user is prompted for a command. Valid commands are:

- U n - Scroll up n transactions. If n is omitted the user will be prompted to enter the desired number. If the user just hits enter the default is 2 transactions.
- D n - Scroll down n transactions. If n is omitted the user will be prompted to enter the desired number. If the user just hits enter the default is 2 transactions.
- F - Display first transaction in the area (oldest data)
- L - Display last transaction in the area (most recent data)
- P - Initiate printing of trace data on a line printer. The user will be prompted to enter A to print the entire trace area, P n to print the last n transactions in the trace area (n = 0 will end display without printing), or EN to end without printing.
- C - Check to see if an overrun error occurred while tracing. For explanation of this option, see the section of this manual dealing with Trace Overrun.

If, for some reason, the trace facility did not finish writing the desired area successfully the display facility will help the user recover the data. It will display the last known position of the last record written. It is up to the user to locate the last record written and supply the display facility with its physical record number. Since the header area is updated for every 16 records written the last record written can be found within the next 16 records of the last known value. A typical situation would be as follows:

```
** S/1 CRASHED BEFORE FINAL HEADER UPDATE **
NOTE THE NUMBER OF THE LAST RECORD WRITTEN
SO THAT THE DISKETTE HEADER CAN BE PATCHED
TRACE AREA START RECORD: 3      END RECORD: 259
LAST KNOWN RECORD WRITTEN: 163
NO WRAP OCCURRED
DISPLAY STARTING AT RECORD: 163
HOW MANY RECORDS TO DISPLAY: 5
```

The desired data will be displayed and the user can find the last record written by examining the time stamps for either a decrease or a large increase. Notice that only five records were requested. This way if the last

IBM Internal Use Only

record is found the user won't have to scroll through reams of data before updating the header. If the last record isn't found you can display more data by replying Y to the following prompt.

```
DISPLAY MORE? N
DO YOU WANT TO PATCH THE DISKETTE? N - terminate display
                                      Y - continue
DO YOU REMEMBER THE NUMBER OF THE LAST RECORD WRITTEN? N - redisplay
                                                         Y - continue
ENTER IT: 167
167 IS LAST RECORD WRITTEN? Y
DISK HEADER UPDATED...TRY DISPLAY AGAIN
```

At this point the user can reissue the DISPLAY command and data will be displayed correctly.

H.2.3 Trace Overruns

When transmission rates between the DSC and the S/I are very high, the S/I cannot empty the trace buffer to disk fast enough, and data is lost because of trace overrun.

H.2.3.1 Detection: P and C Options of DISPLAY

If the 'P' option is used and overrun occurred, a message:

```
'D A T A   L O S T'
```

will appear on the printout at the transaction where the error occurred. Note, the message will not appear on the first transaction of a partial print of the trace, regardless of whether overrun occurred there. The P(Print A)ll command is recommended for detection using 'P'.

If the 'C' option is used, the program will search through the entire trace from beginning to end and list on the screen the transaction numbers where overrun occurred. The messages will be:

```
'ERROR IN TRANSACTION: x OF y. nnnnnnnn (HEX) MICROSECONDS LOST.'
'...CHECK COMPLETE'
```

If no only

```
'...CHECK COMPLETE'
```

message appears, no overrun occurred.

H.2.3.2 Dealing with the Problem

Change in buffer size - Two versions of the trace program exist: one called TRACEB (Trace Big) and the other TRACES (Trace Small). TRACEB's trace buffer holds 128 transactions as compared to TRACES's 32. Overrun is much less likely to occur using TRACEB. Use of this program is recommended over TRACES, even though it uses considerably more memory.

Change of baud rate - The higher the baud rate used, the more likely overrun is to occur. To minimize the overruns, keep the rate low.

Use of the hard disk - If overrun is a serious problem, dumping to the hard disk should be considered, since it is on the order of ten times faster than the EDXDMP floppy. At the beginning of the trace program, the user is asked to enter a volume name or press 'enter'. Pressing 'enter' selects the EDXDMP floppy diskette; typing a valid volume name selects the hard disk.

IBM Internal Use Only

Be sure, that the 'DUMP' dataset has been allocated on the volume you select before trying to trace to the hard disk.

Program	Baud	Disk	Possiblility
TRACES	19.2kB	Floppy	Possible under extreme usage
		Hard	Unlikely
	38.4kB	Floppy	Likely under extreme usage
		Hard	Unlikely
TRACEB	76.8kB	Floppy	Likely
		Hard	Possible
	19.2kB	Floppy	Unlikely
		Hard	Unlikely
	38.4kB	Floppy	Possible under extreme usage
		Hard	Unlikely
	76.8kB	Floppy	Likely
		Hard	Possible

Figure 67. TRACES/TRACER Overrun

IBM Internal Use Only

IBM Internal Use Only

APPENDIX I. S/1 COMPARE DATA SET UTILITY - \$COMP

This document outlines the use of the S/1 compare utility (\$COMP).

The compare utility was developed to show differences between two versions of a load module. The files can reside on any volume, not necessarily the same one. It gives the user the option to compare the entire files or just selected portions. For a partial compare the user can specify the start address for each file thus allowing the user to align the two files being compared. Also, the user can choose to display results at the terminal or on a printer.

IBM Internal Use Only

I.1 USING \$COMP

There are six commands currently supported by the compare utility. In the following sections these commands are described. Prompts from the program and suggested user responses are given. The user responses are highlighted.

I.1.1 Valid commands

The commands currently supported by \$COMP are:

? - display list of valid commands

NF - (New Files) lets user identify the files to be compared.

CD - (Change Destination) lets the user specify the output destination.

CA - (Compare All) program will compare all records until end of file is reached in one of the modules.

CP - (Compare Partial) program will compare, starting at a user specified offset in each file.

EN - terminate the compare program

I.1.1.1 The NF (New Files) command

To identify to the program the files to be compared use the NF command. The following is the sequence of prompts and responses if T99FILE1 on EDX003 and T99FILE2 on EDX002 were to be compared.

```
COMMAND (?): NF
NAME OF FIRST FILE: T99FILE1
EDX003 OK FOR T99FILE1? Y
SEARCHING FOR T99FILE1.....
NAME OF SECOND FILE: T99FILE2
EDX003 OK FOR T99FILE2? N
VOLUME FOR T99FILE1: EDX002
SEARCHING FOR T99FILE2.....
    14 RECORDS IN T99FILE1 ON VOLUME EDX003
    16 RECORDS IN T99FILE2 ON VOLUME EDX002
COMMAND (?):
```

I.1.1.2 The CD (Change Destination) command

To query or change the output destination issue the CD command. The following is what will be displayed when this command is entered.

IBM Internal Use Only

COMMAND (?): CD

CURRENT OUTPUT DESTINATION: TERMINAL

P - PRINTER OUTPUT

T - TERMINAL OUTPUT

X - PRESENT DESTINATION

EN - END COMPARE ROUTINE

NEW DESTINATION: P

COMMAND (?):

At this point the output will be routed to the printer. The valid destinations are self explanatory and need no elaboration.

I.1.1.3 The CA (Compare All) command

If you want to compare two files from start to finish, both beginning at offset zero, use the CA command. When you have issued this command you are not locked in until end of file is reached. Whenever the screen has filled with data you are prompted for a command. To terminate the CA command enter X. To end the compare program enter EN. To scroll through the files just hit ENTER.

Data is displayed as follows,

FILE1		FILE2	
ADDRESS	CONTENTS	CONTENTS	ADDRESS

The address in each file is displayed because the same display routine is used as for CP (compare partial) where the addresses within the two files aren't necessarily the same.

I.1.1.4 The CP (Compare Partial) command

If you want to compare two files but not necessarily from start to finish and/or not necessarily from address zero the CP command is issued. Again, you are not locked in until the end is reached. Whenever the screen has filled with data you are prompted for a command. To terminate the CP command enter X. To end the compare program enter EN.

To change the scroll direction from down to up enter U, and to change from up to down enter D. To continue scrolling in the present direction just hit ENTER. Data is displayed as follows,

FILE1		FILE2	
ADDRESS	CONTENTS	CONTENTS	ADDRESS

The address in each file is displayed because the start address in each file is user specified. Therefore they aren't necessarily the same. In the following example it is desired to compare T99FILE1 with T99FILE2 starting address X'0500' and X'0201' respectively.

IBM Internal Use Only

COMMAND (?): CP

START ADDRESS FOR T99FILE1 (WORD) : 500

START ADDRESS FOR T99FILE2 (WORD) : 201

T99FILE1		T99FILE2	
ADDRESS	CONTENTS	CONTENTS	ADDRESS
500	000C	00A0	201
520	F72C	435F	221
:	:	:	:
:	:	:	:

CMD: X

n DIFFERENCES FOUND

COMMAND (?):

IBM Internal Use Only

APPENDIX J. DISK UTILITY PROGRAM (DUT)

This document is intended to provide the user with a basic understanding of the disk utility. Available through the disk utility, are three types of commands: dataset oriented commands, file oriented commands, and general purpose commands. The inexperienced user should find the prompts provided by the disk utility sufficient to lead him through the various available functions.

Upon loading of the disk utility, the user is presented with a menu of available commands. To execute a desired function, the user must enter the desired command on the command prompt line. The user will then be prompted to enter additional information that the selected function requires. As the user becomes more experienced, he may in most cases, pre-enter information required by a given function.

IBM Internal Use Only

J.1 DATASET QUALIFICATION

Datasets may reside on up to four disk drives depending on the configuration of the DSC being used. The disk drives, themselves, are identified by drive numbers, whereas disks or diskettes are identified by volume identifications. Datasets may be qualified at any time, by appending to the dataset name being specified, a comma followed by either a drive number or a volume identification. The following examples illustrate two ways of qualifying the dataset TEMP which resides on a diskette with volume id EDX001 inserted in disk drive #1.

- TEMP,EDX001
- TEMP,1

Note, that this dataset qualification facility prohibits the use of a comma as part of a dataset name. Some disk utility functions do not require any dataset qualification. In these cases, drives will be searched in order for the first occurrence of the specified dataset. Other disk utility functions require qualification as to where a user specified dataset resides. In these cases, if the user does not qualify the dataset, he will be prompted to enter the drive number on which the specified dataset resides. A 0 drive number entered at this point acts as an escape, causing the option menu to be redisplayed. If an invalid drive number is specified, the user will be presented with a table indicating the volume identifications of the diskettes that are currently in each of the disk drives. If a disk drive is empty, the corresponding volume identification will be indicated by asterisks. This table is intended to assist the user in specifying the correct drive number. Note that the same table may be displayed at any time by issuing the 'V0' (vary on display) command.

J.2 DATASET ORIENTED COMMANDS

The following commands provide dataset manipulation capabilities:

- CD - Compare datasets. This function allows the user to compare the contents of two datasets.
- AL - Allocate dataset. This function allows the user to create a new dataset.
- CP - Copy dataset. This function allows the user to copy a dataset from one drive to another.
- CA - Copy all datasets. This function allows the user to copy all datasets from one disk file to another.
- CS - Copy S/l dataset. This function allows the user to copy a S/l dataset to a disk file.
- DE - Delete dataset. This function allows the user to delete a dataset from a disk file.
- DU - Dump dataset. This function allows the user to display the contents of a dataset on the screen.
- PA - Patch dataset. This function allows the user to patch a dataset.
- MV - Move dataset. This function allows the user to move a dataset from one disk file to another.
- RE - Rename dataset. This function allows the user to rename a dataset.
- RH - Read dataset header. This function allows the user to display the header of a dataset.

IBM Internal Use Only

J.3 FILE ORIENTED COMMANDS

The following commands provide disk file manipulation capabilities:

CM - Compress disk file. This function allows the user to compress a disk file. In order to use this function, the user must provide a second disk file that contains enough contiguous free space to accommodate the the largest dataset on the disk file being compressed. Throughout the execution of the compress function, a scratch dataset called 'DUT-TEMP' will be created and deleted as each dataset is relocated on the disk file being compressed. Thus, the user must ensure that none of his datasets have the name 'DUT-TEMP'. Note that the second diskette will be left as it was upon completion of the compress function.

FM - Format disk file. This functions allows the user to format the following types of disk files:

1. type 1 diskettes,
2. type 2 diskettes,
3. type 2D diskettes, and
4. hard files.

The following information is required of the user in order to format and initialize a disk file:

drive # - refers to the number of the drive in which the disk file to be formatted resides (0 will allow the user to escape this function).

type refers to the type of disk file (1,2,2D,HF).

format - refers to the sector byte density (128 or 256). Note that hard files are automatically formatted with 256 bytes per sector.

volume id - refers to a user specified volume identification up to six characters in length. Note that the first character of the volume identification must not be a decimal digit.

owner id - refers to a user specified owner identification up to twelve characters in length.

directory size - refers to the number of members to be included in the disk file directory.

RV - Read volume/directory. This function displays volume and the directory information pertaining to the disk file which currently resides in the user specified disk drive.

RF - Read free space entries This function displays free space entry information pertaining to the disk file which currently resides in the user specified disk drive.

RD - Read a dataset or a physical location on a disk file. This function allows the user to dump to memory either a dataset or a physically addressable section of memory. In each case, the specified data will be displayed one record at a time. Each record displayed will be stored in memory at the indicated data start address. Addresses to the left of the data which are displayed will be relative to this data start address. To read a physical location of a disk file, the user must use one of the following reserved dataset names:

- \$\$DRx
- \$\$VLx

The x refers to the desired drive number (1 through 4). \$\$DRx allows the user to access diskette files from cylinder 1 on, and hard files from cylinder 0, head 1 on. \$\$VLx allows the user to access an entire disk file. If the user enters one of these reserved dataset names, he will be prompted to enter a physical address. The entered physical address must be in the form CCHHRR, where:

IBM Internal Use Only

CC refers to the cylinder,

HH refers to the head, and

RR refers to the record.

At least 5 digits comprising a physical address must be entered. For users familiar with disk file organization, this command can be used in conjunction with the WR (write to disk) command in order to patch disk files.

WR - Write to a disk file. This command allows the user to write from memory to a specified physical address on a disk file. The memory address from which data will be written can be user specified or defaulted to the address displayed by the last read (RD) command. The following is an example of how a disk could be patched using the RD and WR commands:

1. read, using the RD command, the area on disk to be patched (use a dataset name of either `$$VLX` or `$$DRx`).
2. note the memory address at which the area on disk is stored.
3. hit the PF7 key to enter Debug.
4. using the memory display and patch commands, alter the data placed in memory by the RD command.
5. quit debug, thereby returning to the disk utility application.
6. issue the write (WR) command, and specify the same dataset name and physical address as was specified in the RD command.
7. accept the displayed default address by hitting enter when prompted to enter a start address.
8. the disk utility program will then write the patched area of memory to disk.

This command is intended for users familiar with disk file organization who wish to patch non-dataset areas of a disk file.

VO - Vary on display. This command displays to the user the volume identifications of the disk files currently in the disk drives. An empty disk drive is indicated by a row of asterisks in place of the volume identification field. It also displays a table indicating the types and formats of the disks that reside in each of the four drives.

IBM Internal Use Only

J.4 GENERAL PURPOSE COMMANDS

The following commands provide general purpose functions:

- SM - Scroll memory. This command allows the user to display a specified section of HAZEL memory.
- QU - Make the disk utility program quiescent. To activate the program from its quiescent state use the DUT attention command.
- EN - End the disk utility program.

IBM Internal Use Only

APPENDIX K. S/1 AND DSC UTILITY PROGRAMS

The following is a list of various utility programs and their description:

- \$DSCPRT** - This S/1 program is required by the \$PRT Supervisor Utility Function (see "Supervisor Utility Functions" on page 173). It must be recorded in the S/1 Big Table.
- SIDEBUG** - This S/1 program allows some of the Supervisor Utility Functions to be executed from S/1. See "Remote Support" on page 176.
- TRACEB** - This S/1 program allows to trace TP transactions from S/1. See "Communications Trace - S/1" on page 214.
- TRACES** - This S/1 program allows to trace TP transactions from S/1. See "Communications Trace - S/1" on page 214.
- DEBUG** - This DSC program provides debug facilities for AP/DISK. The main use for AP/CSS is the RAM nucleus load function.
- DEEGA** - This DSC program provides debug facilities for AP/DIS. The main use for AP/CSS is the RAM nucleus load function. This program is the Ariel version of the DEBUG program.
- \$PATCH** - This S/1 program allows to patch S/1 and DSC programs on any S/1 volume.
- QLINE** - This S/1 program, on request, sends a message to a non-existent program in a specified node. A return code of 5000 is expected.
- QALL** - This S/1 program sends a message to a non-existent program in all nodes. A return code of 5000 is expected.
- UTDH** - This S/1 program allows to view a DSC memory dump stored on a diskette.
- DTD** - This S/1 program receives memory dump from a DSC and stores it on a diskette.
- DUT** - This DSC program provides various disk utilities for a tool controller with the IBM disk (see "DISK" on page 220). For the description of the DUT program see "Appendix J. DISK UTILITY PROGRAM (DUT)" on page 339.
- \$COMP** - This S/1 program shows differences between to data sets. See "Appendix I. S/1 COMPARE DATA SET UTILITY - \$COMP" on page 335.
- EDXJ4** - This DSC program generates EDX instruction times for most EDX instructions.
- DIS** - This DSC program allows the user to perform DIS I/O operations for a selected block and macrofunction.
- DISAP** - This DSC program is the ARIEL version of the DIS program.
- DSCDUMP** - A CLIST to run the dump formatting program (FRMTDUMP) on S/370.
- FRMTDUMP** - A PL/I program that formats the DSC dump.
- LOAD1** - A DSC program for Hazel/4 to generate disk load statistics.
- LOAD3** - A DSC program for Hazel/3 to generate host communications error statistics.
- LOAD3** - A DSC program for Hazel/4 to generate host communications error statistics.
- LOAD8** - A DSC program for Hazel/Ariel to generate host communications error statistics.

IBM Internal Use Only

INDEX

\$ 12, 37
 \$? 173
 \$A 173
 \$BDCHD 253
 \$BHXHD 253
 \$BF 173, 223
 \$C 173
 \$CHAINP 242
 \$CMSETUP 227
 \$CP 173
 \$D 173
 \$DCBWD 253
 \$DEQT 252
 \$DIS 173
 \$DPYPROC 254, 255
 \$ENDATTN 252
 \$ENQT 252
 \$FREEMAIN 250
 \$GETMAIN 250
 \$GETPAR3 227
 \$GVALUE 252
 \$GVALU4 252
 \$HXBWD 253
 \$L 175
 \$LH 175
 \$LHNG 175
 \$LNG 175
 \$L5KYWT 254, 255
 \$NOVIDEO 28
 \$P 175
 \$PARMx 102, 104, 131
 \$PRNTEXT 252
 \$PRNTNUM 252
 \$PRNTNUM4 252
 \$PROGORG 250
 \$PRTOUT 257
 \$QUESTN 252
 \$READTXL 252
 \$READTXT 252
 \$S 175
 \$SCRNMGR 254
 \$SKLNSP 252, 257
 \$SMAPAVL 250
 \$SYSCOM 215, 216
 \$T 89, 175
 \$TKYWSH 252
 \$TMICHAN 248
 \$TMIKEY 248
 \$TMIVALU 248
 \$VALSCAN 253
 \$W 175
 \$X 175
 *EJECT 67
 *MODLISTING 13, 32
 *MODLISTING END 13, 32
 *SPACE 152
 > 28, 252
 # 12
 #PRGTCB 237
 #Rx 165
 #TCBPTR 237
 #TCBTCH 248
 @ 12, 126, 162

A

A - address 47
 active 143, 231, 234, 238, 240, 252, 258
 ADD 18, 20, 33
 addition 18, 20, 22
 ADDRESS 218
 address - A 47
 address
 12 of current TCB
 ADDV 34
 advanced input 28, 90, 139
 Analog Input card (AI) 107, 109
 Analog Output card (AO) 107, 109
 AND 18, 20
 AND - EDX command 35
 AND - in relational statement 26
 AP 196, 251, 254, 257, 270, 273
 AP Device Table (APDVTB) 258
 AP/CSS 196, 205, 228, 257, 273, 288, 345
 assembly date 205
 Bring Up Tests 273
 Console Support 257
 hardware level assignment 228
 interrupt levels 228
 IPL 196, 288
 machine check 205
 RAM nucleus 345
 Run Error Light 273
 AP/DIS 205, 228, 273
 assembly date 205
 Bring Up Tests 273
 hardware level assignment 228
 interrupt levels 228
 machine check 205
 Run Error Light 273
 AP/DISK 196, 205, 220, 228, 273, 288
 assembly date 205
 Bring Up Tests 273
 hardware level assignment 228
 interrupt levels 228
 IPL 196, 288
 machine check 205
 Run Error Light 273
 APDVTB 205
 ARIEL 1, 3, 11, 53, 60, 96, 105, 109, 112, 228, 270, 272, 273, 288, 296, 311, 320
 Bring Up Tests 270, 273
 compatibility 311
 hardware level assignment 228
 Initialization 272
 interrupt levels 228
 IPL 288
 LOADH return code 105
 memory dump to S/1 320
 Nucleus Directory 296
 Run Error Light 270, 272
 ASCII 306
 assembly date 205
 ATTACH 36, 52, 133, 231, 238, 240, 244, 247
 Attached Processor (AP) 251, 254, 257, 270
 ATTACHI 247

IBM Internal Use Only

Attention Key 28, 37, 173, 252
ATTNLIST 28, 37, 70

B

B 165
BAL 165
BALL 165
BALLR 165
BIT 95, 96, 107, 109
BIT 95, 96
BITS= 54, 57, 60, 107, 109, 112
Blast Reset 280
BLOCK 95, 96
Block Interface Card (BIC) 95, 96, 107, 109
BLOCK= 107, 109
Bring Up Tests 270, 273
BRINGUP 222, 270
Bring Up Tests 270
BUFFER 38, 97
Buffer Management 253
BUSY 78, 239
BUSY= 77

C

C - character 47
CALL 39
CANCEL 40
CARDS 219
CCB 238, 251, 252, 260, 302
Equates 302
CCB WAIT chain 252
Central Processing Unit (CPU) 219
chain field 66, 235, 236
ECB 66, 235
QCB 236
chaining 76, 112, 181, 198, 212, 231, 235, 236, 237, 238, 239, 240, 242, 248, 252, 257, 258, 265, 312, 314
\$CHAINP supervisor call 242
APDVTB chain 258
CHAIN parameter on MDCB 112
chain pointer in QCB 76
chain to the Ready Queue EDX
Trace entry 181, 198
DCB chaining 257
ECB chain 235
MDCB chain 265
MDCB chaining 312, 314
QCB chain 236
Ready Queue 231, 238, 239, 240
Send Chain Queue in Communications Trace 212
TCB chain 237
character - C 47
CHARDEL 225
CHKKEY 41
CLOSE 30, 42, 121
code 36, 49, 52, 123, 133, 238
ATTACH 36, 238
DEQ 49
DETACH 52, 238
POST 123, 133, 238
code field 66, 235, 236, 238, 239
ECB 66, 235, 238

QCB 236, 239
CODE= 36
COMM 215, 218, 221
command service routine (CSR) 248
Command Tag Code (CTC) 54, 57, 60, 112, 311
command word 227
common logarithm 22
Common Setup Routine (SCMSETUP) 227
COMSIZE 216
Console Control Block (CCB) 238, 251, 252, 260
CONVIB 43
CONVID 45
COPY 165, 215, 221
count 18, 20, 38, 54, 55, 57, 58, 60, 63, 85, 88, 90, 112, 141, 148, 154
ADD 18, 20
AND 18, 20
BUFFER 38
DISIO 112
DISREAD 54, 55, 60
DISWRITE 57, 58, 60
DIVIDE 18, 20
DO 63
EOR 18, 20
FPCONV 85
GETMAIN 88
GETVALUE 90
IOR 18, 20
MDCB 112
MOVE 18, 20
RECEIVE 141
SEND 148
SHIFTL 18, 20
SHIFTR 18, 20
STIMER 154
SUBTRACT 18, 20
COUNT= 54, 57, 60, 112
CPU 219
CRDELAY 225
CSR 248
CSS/AP 251
Console Support 251
CTC 54, 57, 60, 112, 266, 267, 311
CTC= 54, 57, 60, 112, 311

D

D - decimal constant 47
DATA 47, 86, 102, 104, 125, 139
Data Control Block (DCB) 257
data definition 47
A - address 47
C - character 47
D - decimal constant 47
E - standard floating point 47
F - decimal constant 47
L - extended floating point 47
X - hexadecimal constant 47
Data Manipulation 15
DATA= 148
DATE 89
DCB 257
decimal constant - D 47
decimal constant - F 47
DEQ 49, 216, 231, 239, 240, 244, 245, 247
DEQIA 247

IBM Internal Use Only

DEQT 28, 37, 50, 78, 251, 252
 derived floating point 22
 DEST 51, 148
 DETACH 37, 52, 231, 238, 240, 244
 detached 131, 234
 DEVICE 225
 DEVICE= 139
 Digital Input card (DI) 107, 109
 Digital Output card (DO) 107, 109
 DIS 1, 5, 53, 54, 57, 60, 107,
 109, 112, 113, 114, 123, 173, 221,
 265, 311
 Error Log 114
 Internal Commands 113
 Internal DIS Commands 112, 173
 DISIO 17, 53, 265, 311
 serial 265
 disk 30, 65, 102, 137, 170, 175,
 196, 205, 220, 228, 273, 288, 307,
 339, 345
 \$L Supervisor Utility
 Function 175
 assembly date 205
 Bring Up Tests 273
 DISK command 220
 Disk Directory Layout 307
 disk functions 30
 Disk Utility Program (DUT) 339,
 345
 DSCB 65
 hardware level assignment 228
 interrupt levels 228
 IPL 196, 288
 LOAD command 102
 machine check 205
 Run Error Light 273
 WRITE 170
 Display Processor (\$DPYPROC) 255
 DISREAD 17, 54, 60, 265, 311
 Distributed Interface System
 (DIS) 5, 53, 54, 57, 60, 112,
 123, 221, 265, 311
 Distributed System Controller
 (DSC) 1, 5, 8, 9, 30, 102, 104,
 224
 DISWRITE 17, 57, 60, 143, 167,
 265, 311
 DIVIDE 18, 20, 62
 division 18, 20, 22
 DO 63, 71
 double precision 18, 19, 26, 28,
 43, 45, 85, 90, 91, 97, 129, 252,
 253
 DSC 1, 5, 8, 9, 30, 102, 104, 224
 DSCB 65

ELSE 26, 68, 72, 93
 END 11, 12, 13, 32, 69, 215
 END ECB 52, 75, 160
 end of data 137, 170
 END= 137, 170
 ENDATTN 37, 70, 252
 ENDDO 71
 ENDDIF 26, 68, 72
 ENDPORG 11, 12, 30, 73, 76, 160,
 265
 ENDSELECT 74, 122, 145, 168
 ENDSYS 215, 221
 ENDTASK 37, 52, 75, 160
 ENDTP 30, 76
 ENQ 49, 77, 216, 231, 239, 240,
 246
 ENQT 28, 30, 37, 50, 78, 251, 252
 ENVIR 223
 EOR 18, 20, 79
 EPROM card 288
 EQ 26
 EQU 80

Equate Tables
 CCB 302
 IDB 304
 MDB 304
 MDCB 303
 Nucleus Directory -
 Hazel/Ariel 296
 Nucleus Directory - Hazel/3 293
 Nucleus Directory - Hazel/4 with
 AP/CSS 295
 Nucleus Directory - Hazel/4
 without AP/CSS 294
 Program Header 304
 ICB 301
 ERROR 54, 57, 60, 86, 88
 ERROR= 53, 54, 57, 60, 102, 104,
 137, 170
 event 123, 143, 167
 Event Control Block (ECB) 66, 95,
 96, 123, 143, 148, 216, 235, 238,
 258
 EVENT= 53, 60, 102, 104, 131, 148,
 160
 exclusive or 18, 79
 EXEC 102, 104
 execution state
 active 143, 231, 234, 238, 240,
 252, 258
 detached 131, 234
 ready 234
 waiting 234, 240, 252, 258
 EXP 22, 81
 Exponential 22
 extended floating point - L 47

E

E - standard floating point 47
 EBCDIC 28, 43, 45, 162, 253, 265,
 305
 ECB 66, 95, 96, 123, 143, 148,
 159, 216, 235, 238, 258
 chain field 66, 235
 code field 66, 148, 235, 238
 key field 66, 235
 validation field 66, 235
 ECHO 225
 EJECT 67, 164
 Electrostatic Diaphragm Switch card
 (EDS) 107, 109

F

F - decimal constant 47, 91
 FADD 22, 82
 FDIVD 22, 83
 FKEY 20, 118
 FLOAT 26
 floating point 22, 43, 45
 FMULT 22, 84
 FORMAT 43, 45
 FORMAT= 90, 129
 FPCONV 85
 FREEMAIN 86, 88
 FSUB 22, 87

IBM Internal Use Only

G

GE 26
 GEN 125
 GETMAIN 88
 GETTIME 89
 GETVALUE 28, 90
 GOTO 26, 52, 92, 93
 computed 92
 GT 26

H

hardware level assignment 228
 Hazel 228
 hardware level assignment 228
 interrupt levels 228
 HDWRTST 215, 222
 Header 102, 104, 216, 237, 257,
 286, 304
 DCB 257
 Program 102, 104, 216, 237
 Program Equates 304
 Response Block 286
 hexadecimal constant - X 47
 HOSTCOMM 221

I

IDB 95, 96, 265, 269, 286, 304,
 311
 Equate 304
 Layout of 269
 Idle Loop 231, 240
 IF 26, 68, 72, 93
 Immediate Action - (IA) 247
 INDEX 38, 63, 92, 97
 index register 14, 18, 23, 227,
 228
 Initial Program Load (IPL) 288
 Initialization 272
 INIBIT 11, 17, 95, 96, 123, 143,
 167, 218, 221, 265, 311
 Internal DIS Commands 112
 interpreter 6, 240
 Interrupt Data Block (IDB) 95, 96,
 265, 286
 interrupt handler 252, 254, 260
 Level 2 252, 254
 Level 4 252, 260
 interrupt levels 228
 Interval Timer chain 248
 interval timer interrupt service
 routine (ISR) 248
 Interval Timer Queue 248
 INTIME 38, 97
 INIx 95, 96, 123
 IOCB 78, 98
 IOCB= 50
 IOR 18, 20, 99
 IPL 97, 196, 288
 ISR 248

J

JIBPMAC 165

K

key field 36, 40, 49, 66, 77, 88,
 102, 104, 141, 148, 167, 235, 236,
 286
 ECB 66, 235
 QCB 236
 KEY= 36, 40, 49, 77, 86, 88, 102,
 104, 123, 137, 141, 143, 148, 167,
 170
 Keyboard card (KEYBD) 107, 109
 Keyboard Wait Support
 (\$L5KYWT) 255

L

L - extended floating point 47
 LAD 165
 LE 26
 LEAVE 100, 145, 168
 LENGTH 162
 Level 2 interrupt handler 252, 254
 Level 4 interrupt handler 252, 260
 LINE= 90, 126, 129, 139
 LINEDEL 225
 LINSIZE 225
 LN 22, 101
 LOAD 102
 LOADH 30, 37, 104, 131, 285
 LOADNUC= 102, 104
 loadpoint 86
 LOADPT= 102, 104, 286
 LOADREG 242
 loc 137, 170
 LOG 22, 106
 Logical Space Address (LSA) 54,
 57, 60, 107, 109, 112, 218, 221,
 265, 311
 LOGMSG= 42, 102, 104, 133
 Longitudinal Redundancy Check
 (LRC) 222
 LRC 222
 LRCHECK 222
 LSA 11, 17, 54, 57, 60, 107, 109,
 112, 218, 221, 265, 311
 LSA= 311
 LSAX 54, 57, 60, 107, 109
 LT 26

M

machine check 36, 40, 55, 58, 118,
 205
 MACRO 107, 109
 macro layout 285, 287
 LOADH 285
 OPEN 287
 RECEIVE 287
 SEND 287

IBM Internal Use Only

Macrofunction Data Block (MDB) 54,
57, 221, 265, 286
Macrofunction Data Control Block
(MDCB) 53, 60, 107, 109, 112,
265, 311
Magnet Driver card (MD) 107, 109
maintask 131, 160, 286
MAFAD 111
MAXPROG 224
MDB 54, 57, 221, 265, 266, 286,
304, 311
 Equates 304
 Layout of 266
MDCB 11, 17, 53, 60, 107, 109,
112, 265, 267, 303, 311
 description of 112
 Equates 303
 Layout of 267
Mini Debug 177
MODE 162
MODE= 90, 126, 129, 139
MODLISTING 13, 32, 215
MODLISTING END 13, 32, 215
MOVE 18, 20, 118
move data 18
 cross partition 18
MOVEA 18, 20, 119
multiplication 18, 20, 22
MULTIPLY 18, 20, 120

N

native display 251, 254
natural logarithm 22
NE 26
NHIST 225
NHIST= 98, 126
NO 136
NODATA 125
NODE= 51
NOGEN 125
NOWORK= 141
Nucleus Directory 293
 Equates - Hazel/Ariel 296
 Equates - Hazel/3 293
 Equates - Hazel/4 with
 AP/CSS 295
 Equates - Hazel/4 without
 AP/CSS 294
NUCTYPE 223

O

OCB 76
OFF 125
ON 125
OPEN 30, 76, 121, 287
Open Control Block (OCB) 76
Operation Monitor card
 (OPMON) 107, 109
OR - EDX command (EOR) 20, 79
OR - EDX command (IOR) 20, 99
OR - in relational statement 26
ORG= 141

ORIGIN 141
OTHER 122, 145, 168
overflow 20, 22
OVFLINE 225
OVFLINE= 98
owner field 236, 239
 QCB 236, 239

P

P= 165
PAGSIZE 225
PAGSIZE= 98, 126
PARM 165
PARM = 131
PARMNAME= 102, 104
partition number 88
parx 39, 155
PAL key 28, 37, 173
PF key 260
PF1 185
PF7 key 28, 37, 173
Photo Amplifier card (PA) 107, 109
PORT= 141, 148
PORTS 51, 121
POST 123, 231, 238, 240, 244, 245,
247
post-processor 13, 32, 67, 125
POSTIA 247
PREC 18, 43, 45, 85
PREC= 137, 170
precision 18, 19, 20, 26, 28, 43,
45, 85, 90, 91, 97, 129, 165, 252,
253
 double 18, 19, 26, 28, 43, 45,
 85, 90, 91, 97, 129, 252, 253
 mixed 18
 single 18, 19, 20, 26, 28, 43,
 45, 85, 90, 91, 97, 129, 165,
 252, 253
PRINDATE 124
PRINT 13, 32, 125
PRINTTEXT 126, 162
PRINTIME 128
PRINTNUM 126, 129
PRIOR= 148
priority 6, 25, 36, 55, 58, 60,
112, 131, 160
process check 36, 40, 55, 58, 118,
205
PROGRAM 11, 102, 104, 131, 133,
160, 216, 221
Program Header 102, 104, 216, 237
Program Header Equates 304
Program Level Check 286
PROGSTOP 12, 30, 37, 42, 133
PROMPT 28
PROMPT= 90, 139
PROTECT= 139
PWR 22, 134
Px= 12, 16, 36, 39, 40, 43, 45,
49, 51, 52, 53, 54, 57, 60, 63,
77, 86, 88, 89, 90, 92, 97, 102,
104, 112, 123, 126, 129, 133, 136,
139, 141, 143, 148, 154, 163, 167,
169
 how to use 16

IBM Internal Use Only

Q

QCB 49, 77, 135, 159, 216, 235,
236, 239
 chain field 135, 236
 code field 135, 236, 239
 key field 135, 236
 owner field 135, 236, 239
 validation field 135, 236
QUESTION 136
Queue Control Block (QCB) 49, 77,
135, 216, 235, 236, 239

R

RAM 225
RAM/Video macrofunction 28, 225,
254
Random Access Memory card
 (RAM) 107, 109
RC= 141, 148
RCNT= 141
RCTC 266, 267
READ 31, 137
Read Only Memory card (ROM) 107,
109
READTEXT 28, 139
ready 234
Ready Queue 231, 238, 239, 240
RECEIVE 30, 141, 281, 287
recnt 137, 170
REGSAVE 242
relative time 97
Relocation Dictionary (RLD) 286
relrec 137, 170
reltime 97
RESET 143, 167
resource 49, 77, 135, 239, 251
 owner 135, 239
 serial 77, 251
 system 77
 user 77
RESULT 18, 22
RETURN 144, 155, 165
return code 65, 81, 82, 83, 84,
87, 101, 106, 153, 169
 AP/CSS Console Support 259
 arithmetic 22, 23
 CANCEL 40
 Console Support 253
 CONVTB 43
 CONVID 45
 DISIO 53
 disk 65
 DISREAD - DSC 55
 DISWRITE - DSC 58
 DISWRITE and DISREAD -
 ARIEL 61
 EXP 81
 FADD 82
 FDIVD 83
 FMULT 84
 FREEMAIN 86
 FSUB 87
 GETMAIN 88
 INTIME 97
 IPL Loader - AP/CSS 289
 LN 101
 LOAD 103

LOADH 105, 285
LOG 106
number conversion 253
RECEIVE 141
SEND 149
SQRT 153
WHEREAS 169
RLD 286
RS232C card (RS232C) 107, 109
Run Error Light 270, 272, 273

S

Screen Manager (\$SCRNMGR) 254
SELECT 74, 100, 122, 145, 168
SEND 30, 42, 51, 76, 148, 281, 287
SETTOD 163
shift left 18, 20
shift right 18, 20
SHIFTL 18, 20, 150
SHIFTR 18, 20, 151
single precision 18, 19, 20, 26,
28, 43, 45, 85, 90, 91, 97, 129,
252, 253
SKIP= 90, 126, 129, 139
Solenoid Driver (SD) 107, 109
SPACE 152
SPACES= 90, 126, 129, 139
SQRT 22, 153
square root 22
standard floating point - E 47
Stepper Motor card (STEPM) 107,
109
TIMER 20, 37, 154, 167
STOP= 42
STORAGE 224
Storage Map 111, 250, 286
SUBROUT 39, 155
SUBTRACT 18, 20, 156
subtraction 18, 20, 22
supervisor 5, 6, 54, 57, 60
Supervisor Utility Functions 28,
173
SVSETUP 242
SYSCDEF 157
SYSCOM 157, 159, 215, 216
SYSCOPEN 157, 159
SYSTEM 215, 221, 224
System Common Area (SYSCOM) 157

T

TAB= 90, 126, 129, 139
task 6, 20, 22, 23, 25, 28, 30,
36, 43, 50, 52, 53, 55, 58, 61,
75, 86, 97, 102, 104, 123, 131,
133, 143, 148, 160, 225, 231, 238,
240, 252, 286
 attention 252
 code field 148
 code word 20, 22, 23, 28, 36,
 43, 53, 55, 58, 61, 86, 97,
 102, 104, 131, 225
 dispatcher 231, 240
 END ECB 75, 160, 238
 execution states 231
 main 131, 160, 286
 priority 6, 25, 36, 131, 160

IBM Internal Use Only

Task Control Block (TCB) 40, 131,
160, 165, 231, 237
taskname 20, 22, 23, 28, 36, 43,
45, 53, 55, 58, 61, 86, 97, 102,
104, 131, 148, 160, 225
TBUF 50, 78, 126, 252
TCB 13, 40, 131, 160, 161, 165,
231, 237, 286, 301
Equates 301
TCBGET 161
TCINT 173, 223
TERMERR= 131, 160
TERMINAL 78, 98, 215, 221, 225,
251
TEST REQ 177
TEXT 90, 126, 139, 162
THEN 93
TIME 89, 163
TIME= 141
TIMEOUT= 167
TIMER 154, 167, 248
Timer card (TIMER) 107, 109
TIMES 63
TITLE 164
TKEY 20, 118
trace 181, 188, 197, 209, 210, 214
Communications - DSC 210
Communications - S/I 214
EDX 181, 188, 197, 209
two's complement 19
TYPE 107, 109
TYPE= 90, 129

U

underflow 22
UNTIL 63
USER 165, 286

V

validation field 66, 235, 236
ECB 66, 235
QCB 236
vector addition 18, 20
VIDEO 225
Video card (VIDEO) 107, 109

W

WAIT 17, 20, 37, 131, 154, 167,
231, 238, 240, 245
WAIT chain 231
WAIT= 137, 170
waiting 234, 240, 252, 258
WHEN 26, 145, 168
WHERE'S 169
WHILE 63
WRITE 170

X

X - hexadecimal constant 47

Y

YES 136

3

3101 28, 37, 225, 251, 288

